CICS Transaction Server for z/OS

CICS での Web サービスの使用

バージョン *5* リリース *5*

CICS Transaction Server for z/OS

CICS での Web サービスの使用

バージョン *5* リリース *5*

注記

本書および本書で紹介する製品をご使用になる前に、 751 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、IBM CICS Transaction Server for z/OS バージョン 5 リリース 5 (製品番号 5655-Y04) 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。

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： CICS Transaction Server for z/OS
Using Web Services with CICS
Version 5 Release 5

発行： 日本アイ・ビー・エム株式会社

担当： トランスレーション・サービス・センター

© Copyright IBM Corporation 2005, 2018.

目次

この **PDF** について v

第 1 章 CICS および Web サービス . . . 1

CICS および SOAP Web サービス 4
SOAP ノード 5
SOAP メッセージおよびアプリケーション・データ構造 9
WSDL およびアプリケーション・データ構造 . . 12
WSDL およびメッセージ交換パターン 14
Web サービスのバインディング・ファイル . . . 16
外部標準 16
SOAP アーキテクチャーおよびメッセージ形式 17
SOAP Web サービス使用の計画立案 29
CICS および JSON Web サービス 31
JSON の Web サービスの概念 32
RESTful JSON Web サービスの概念 35
JSON Web サービス使用の計画立案 37
CICS と z/OS Connect 40
z/OS Connect for CICS の機能 46

第 2 章 CICS での Web サービスの構成 47

Web サービスに応じた CICS システムの構成 . . . 47
Web サービスのための CICS リソース 47
WebSphere MQ トランスポートを使用するための CICS の構成 52
Web サービス・アシスタントと WSRR の間の相互運用性 60
Web サービス・インフラストラクチャーの作成 . . 62
Web サービス・インフラストラクチャー . . . 62
SOAP 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 74
SOAP 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 76
JSON 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 78
非 Java JSON 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 80
z/OS Connect for CICS の構成 81
パイプライン構成ファイル 101
アプリケーション・ハンドラー 159
メッセージ・ハンドラー 161
SOAP メッセージ・ハンドラー 171
パイプラインで使用されるコンテナー 177
Web サービスのための実行時処理 211
Web サービス・トランザクションのサポート 221
バイナリー・データの MTOM/XOP 最適化のサポート 231
Web Services Addressing のサポート 241
SAML のサポート 264

第 3 章 Web サービスの開発 267

JSON Web サービスの作成 267
CICS JSON アシスタント 267
JSON アシスタントを使用した JSON サービス・プロバイダー・アプリケーションの作成 . . 312
JSON Web サービスの制約事項 330
アプリケーション・データおよび XML のマッピングと変換 332
CICS XML 支援機能 332
言語構造からのマッピングの生成 445
XML スキーマからのマッピングの生成 447
アプリケーション・データの XML への変換 449
アプリケーション・データへの XML の変換 450
アプリケーション・データおよび JSON のマッピングと変換 452
CICS JSON アシスタント 453
言語構造からのマッピングの生成 524
JSON スキーマからのマッピングの生成 . . . 526
DFHJSON へのリンクによるアプリケーション・データの JSON への変換 528
TRANSFORM DATATOJSON API コマンドを使用した、アプリケーション・データの JSON への変換 529
DFHJSON へのリンクによる JSON のアプリケーション・データへの変換 530
TRANSFORM JSONTODATA API コマンドを使用した、JSON のアプリケーション・データへの変換 531
JSON Web サービス・アプリケーションの作成 532
SOAP Web サービスの作成 533
CICS Web サービス・アシスタント 534
Web サービス・アシスタントを使用した Web サービス・プロバイダーの作成 624
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 624
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 627
チャネル記述文書の作成 631
生成した Web サービス記述文書のカスタマイズ 633
SOAP 障害の送信 635
Web サービス・アシスタントを使用した Web サービス・リクエスターの作成 637
ツールを使用した Web サービスの作成 640
XML を認識する独自の Web サービス・アプリケーションの作成 641
XML 認識サービス・プロバイダー・アプリケーション 641
XML 認識サービス・リクエスター・アプリケーションの作成 643
Web サービスでの Java の使用 645

© Copyright IBM Corp. 2005, 2018

Axis2 JVM サーバーにおける Java プロバイダー・モードの Web サービスの配置 645
XML を生成および解析する Java Web サービスの作成 648
COBOL インターフェースがある Java Web サービスの作成 648
リクエスター・モードの JAX-WS Web サービスの配置 648
Liberty JVM サーバーでの Java プロバイダー・モードの Web サービスの配置 649
SOAP メッセージの検証 650
アプリケーション提供の無効および初期化されていないデータの処理 651
例 1: 10 進数フィールドの耐障害性 652

第 4 章 Web サービスを保護するためのサポート 655
Web Services Security を実装するための前提条件 656
SOAP Web サービスの保護の計画 657
SOAP メッセージを保護するためのオプション . . 658
Security Token Service を使用した認証 660
Trust クライアント・インターフェース . . . 662
SOAP メッセージへの署名 663
署名アルゴリズム 663
署名された SOAP メッセージの例 663
暗号化された SOAP メッセージの CICS サポート 664
暗号化アルゴリズム 665
暗号化された SOAP メッセージの例 666
Web Services Security に合わせた RACF の構成 666
ID 伝搬のためのプロバイダー・モードの Web サービスの構成 669
Web Services Security に合わせたパイプラインの構成 671
カスタムのセキュリティー・ハンドラーの作成 . . 675
メッセージ・ハンドラーからの Trust クライアントの起動 677
z/OS Connect のセキュリティー 678
z/OS Connect サービスおよび API の許可の構成 679

第 5 章 Web サービスのトラブルシューティング 681
SOAP Web サービスのトラブルシューティング 681
配置エラーの診断 681
サービス・プロバイダーのランタイム・エラーの診断 684
サービス・リクエスターのランタイム・エラーの診断 686
MTOM/XOP エラーの診断 688
データ変換エラーの診断 690
JSON Web サービスのトラブルシューティング 692
JSON デプロイメントの問題のトラブルシューティング 692
JSON アシスタントのトラブルシューティング 694
JSON 要求での問題のトラブルシューティング 695
クライアントに返されるエラー応答 696
JSON アシスタントの戻りコード 697

付録 A. JSON 変換プログラム・リンク可能インターフェース・コンテナー . . . 699
DFHJSON-JSON コンテナー 699
DFHJSON-DATA コンテナー 700
DFHJSON-TRANSFRM コンテナー 700
DFHJSON-JVMSERVR コンテナー 700
DFHJSON-ERROR コンテナー 700
DFHJSON-ERRORMSG コンテナー 701

付録 B. Web サービスのサンプル . . . 703
CICS カタログ・マネージャーの実例アプリケーション 703
ベース・アプリケーション 704
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 707
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 714
実例アプリケーションに対する Web サービス・サポート 716
Web クライアントの構成 731
Web サービス対応アプリケーションの実行 . . 732
実例アプリケーションの配置 733
ベース・アプリケーションのコンポーネント . . 738
ファイル構造と定義 747
JSON のサンプル 749
照会ストリングを使用した HTTP GET 要求の例 749
JSON 本体を使用した HTTP 要求例 750

特記事項 751

索引 757

この PDF について

この PDF では、CICS での Web サービスの使用法について説明します。本書の対象読者は、Web サービスをサポートするように CICS を構成するシステム・プログラマー、および Web サービス環境に配置するアプリケーションを担当するアプリケーション開発者です。CICS TS V5.4 より前は、この PDF は「Web サービス・ガイド」でした。

本書で使用されている用語や表記について詳しくは、IBM Knowledge Center の『CICS 資料で使用されている表記規則および用語』を参照してください。

この PDF の作成日

この PDF は、2018 年 12 月 14 日に作成されました。

© Copyright IBM Corp. 2005, 2018

第 1 章 CICS および Web サービス

CICS は Web サービスをサポートします。

Web サービスとは

Web サービスに備わっているインターフェースは実装の詳細を隠します。これにより、実装場所のハードウェア/ソフトウェア・プラットフォームや、作成に使われたプログラム言語とは独立してこれらを使用できます。この独立性により、Web サービス・ベースのアプリケーションを、さまざまなテクノロジーを活用して疎結合されたコンポーネント指向の実装にすることが可能になります。 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で、または他の Web サービスとともに使用できます。

CICS によりサポートされる Web サービス

CICS® は、SOAP と JavaScript Object Notation (JSON) の 2 つの別個の Web サービス・プロトコルをサポートします。この 2 つのプロトコルの特性と利点はそれぞれ異なっています。

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。 この標準は World Wide Web Consortium (W3C) によって承認されている。

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。

JavaScript Object Notation (JSON)
: JavaScript のオブジェクト・リテラル記法に基づく単純なデータ交換形式。JSON はプログラミング言語に対して中立的ですが、C、C++、C#、Java™、JavaScript、Perl、Python などの言語の規則を使用します。

JSON スキーマ
: 構造を記述し、他の JSON 文書の内容を制限する JavaScript Object Notation 文書。

RESTful
: Representational State Transfer (REST) 制約に準拠するアプリケーションおよびサービスに関するもの。

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

© Copyright IBM Corp. 2005, 2018

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 1.1 の仕様は、Simple Object Access Protocol (SOAP) 1.1 で公開される。

SOAP 1.2 の仕様は以下で公開される。

SOAP Version 1.2 Part 0: Primer

SOAP Version 1.2 Part 1: Messaging Framework

SOAP Version 1.2 Part 2: Adjuncts

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。

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。これらのノードには、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す。

UDDI
: Universal Description, Discovery and Integration を参照。

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 はまた、企業が自社で提供する Web サービスに関する情報を登録して他の企業から検索できるようにする仕様の、一般にアクセス可能な一連の実装でもある。 この仕様は OASIS で公開されている。

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 Services Addressing
: Web Services Addressing (WS-Addressing) は、Web サービスおよびメッセージをアドレッシングするための、トランスポートに依存しないメカニズムを提供する。

  WS-Addressing の仕様は、以下で公開される。

  - Web Services Addressing 1.0 - コア
  - Web Services Addressing 1.0 - SOAP バインディング
  - Web Services Addressing 1.0 - メタデータ
  - Web Services Addressing- Submission

Web Services Atomic Transaction
: 全て実行されるか、全く実行されない (all or nothing) プロパティーを持つアクティビティーの調整に使用される、アトミック・トランザクション調整タイプの定義を提供する仕様。

  この仕様は、OASIS によって公開され、Web Services Atomic Transaction に記載されています。

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。

この仕様は、Web Services Description Language (WSDL) で公開されています。

Web Services Security
メッセージの保全性と機密性を提供する SOAP メッセージの一連の機能強化。 この仕様は、OASIS によって公開され、Web Services Security: SOAP Message Security 1.0 (WS-Security 2004) に記載されています。

WS-Atomic Transaction
Web Services Atomic Transaction を参照。

WS-I Basic Profile
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る。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できる。このプロファイルは Web Services Interoperability Organization (WS-I) によって定義されており、バージョン 1.0 は Web Services Interoperability Organization (WS-I) Basic Profile 1.0 で入手できる。

WSDL
Web サービス記述言語を参照してください。

WSS Web サービス・セキュリティーを参照してください。

XML Extensible Markup Language。

XML の仕様は以下で公開される。

SOAP Version 1.2 Part 0: Primer

SOAP Version 1.2 Part 1: Messaging Framework

SOAP Version 1.2 Part 2: Adjuncts

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。

CICS および SOAP Web サービス

CICS では、Web サービス環境に CICS アプリケーションを配備するための 2 つの異なる方法がサポートされています。 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備を実現できます。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 サービス環境に配備するとき、以下のオプションの中から選ぶことができます。

- 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備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のに役立ちます。

  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開始して、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 サービス記述から開始し、作成するプログラムで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備するために必要な 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、アプリケーション・データと、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流れるメッセージをマップします。

  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の間で変換します。

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。

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 ノードで実行できる操作は次のとおりです。

- 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 ノードは次のタイプのいずれかになります。

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。

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 1 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。

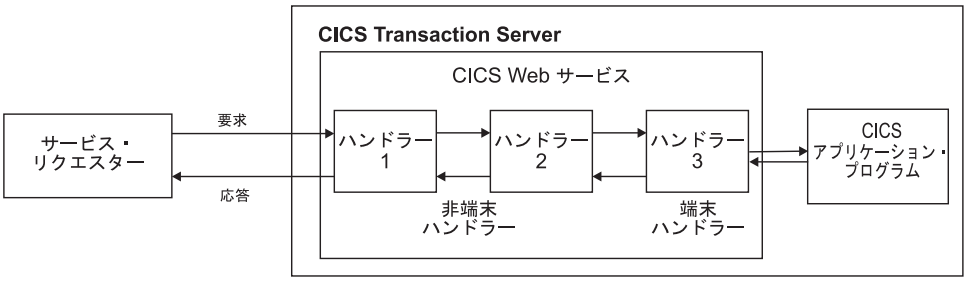


図 *1.* サービス・プロバイダー・パイプライン

1. CICS は、サービス・リクエスタ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 2 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。

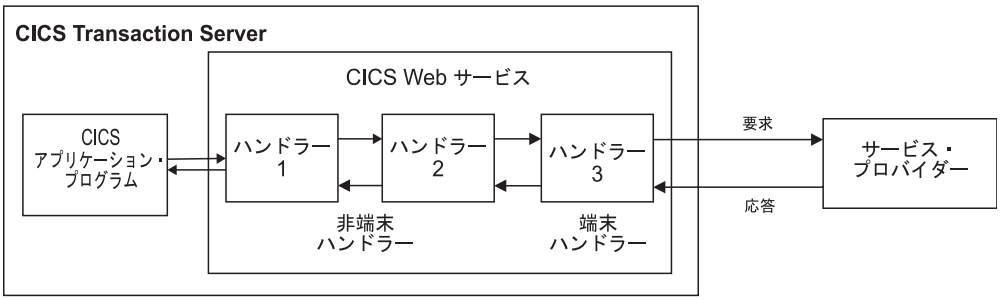


図 2. サービス・リクエスター・パイプライン

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、汎用型です。 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。
- サービス・リクエスター・パイプラインは、SOAP 1.1 または SOAP 1.2 をサポートするように構成できますが、両方をサポートするようには構成できません。ただし、サービス・プロバイダー・パイプラインは、SOAP 1.1 と SOAP 1.2 の両方をサポートするように構成できます。 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。SOAP 1.1 または SOAP 1.2 をサポートするものと、両方をサポートするものがあります。
- 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 バージョン 1.1 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・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。 図 3 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。

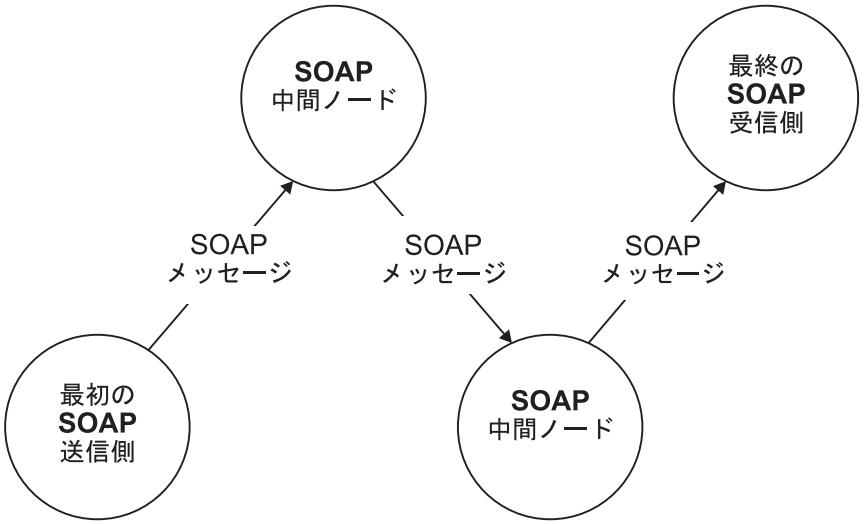


図 *3. SOAP* メッセージ・パスの例

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 本体も処理します。 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 4 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。

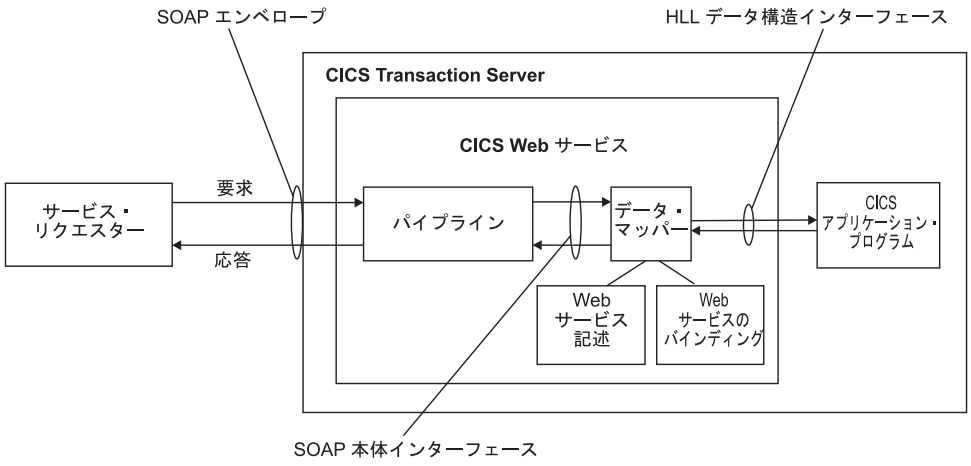


図 *4.* サービス・プロバイダーでの *SOAP* 本体からアプリケーション・データ構造へのマッピング

パイプラインのメッセージ・ハンドラー (一般に、CICS 提供の SOAP メッセージ・ハンドラー) は、インバウンド要求から SOAP エンベロープを除去し、SOAP 本体をデータ・マッパー機能に渡します。 ここでは、SOAP 本体の内容をアプリケーションのデータ構造にマップするために、Web サービス・バインディング・ファ

イルを使用します。 SOAP メッセージの完全な検証がアクティブである場合は、SOAP 本体が Web サービス記述に対して検証されます。アウトバウンド応答がある場合は、処理が反転します。

図 5 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。

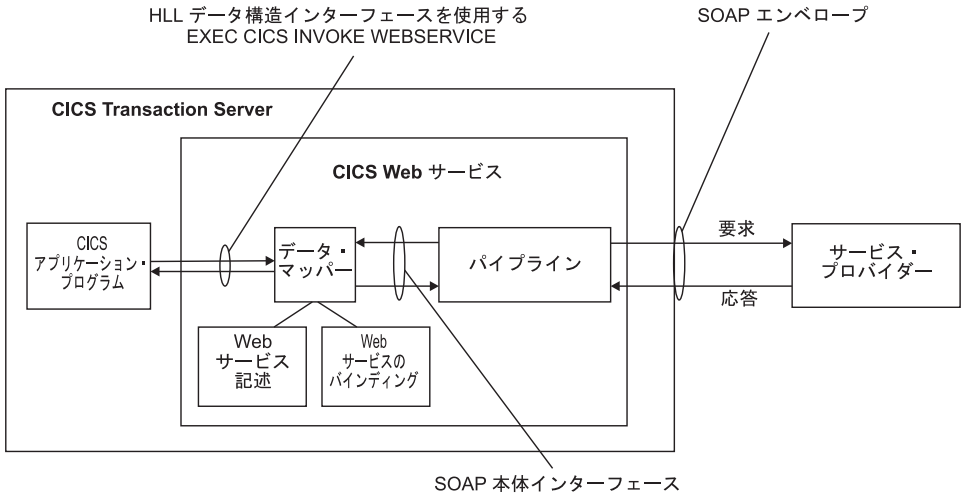


図 *5.* サービス・リクエスターでの *SOAP* 本体からアプリケーション・データ構造へのマッピング

アウトバウンド要求では、データ・マッパー機能が、Web サービス・バインディング・ファイルからの情報を使用して、アプリケーションのデータ構造から SOAP 本体の構成を行います。 パイプラインのメッセージ・ハンドラー (一般に、CICS 提供の SOAP メッセージ・ハンドラー) は、SOAP エンベロープを追加します。 インバウンド応答がある場合は、処理が反転します。 SOAP メッセージの完全な検証がアクティブである場合は、インバウンド SOAP 本体が Web サービス記述に対して検証されます。

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、その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 サービス記述には以下が含まれますが、これ以外にもあります。

- 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 6 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 *6. 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、 14 ページの図 7 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 7. *COBOL* データ構造から導出された *WSDL* フラグメント

WSDL およびメッセージ交換パターン

WSDL 2.0 文書には、SOAP 1.2 メッセージを Web サービス・リクエスターと Web サービス・プロバイダーの間で交換する方法を定義する、メッセージ交換パターン (MEP) が含まれます。

CICS 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・アプリケーションの両方について、 *WSDL 2.0 Part 2: Adjuncts* 仕様および *WSDL 2.0 Part 2: Additional MEPs* 仕様で定義される 8 つのメッセージ交換パターンのうち 4 つをサポートします。 以下の MEP がサポートされています。

In-Only

要求メッセージは Web サービス・プロバイダーに送信されますが、プロバイダーは Web サービス・リクエスターにどのようなタイプの応答を送信することも許可されていません。

- プロバイダー・モードで、CICS が In-Only MEP を使用する Web サービスから要求メッセージを受け取るとき、応答メッセージは戻されません。 DFHNORESPONSE コンテナーが SOAP ハンドラー・チャネルに入れられて、パイプラインが応答メッセージを送信してはならないことが示されます。
- リクエスター・モードでは、CICS は要求メッセージを Web サービス・プロバイダーに送信して、応答を待機しません。

In-Out

要求メッセージが Web サービス・プロバイダーに送信され、応答メッセージが Web サービス・リクエスターに戻ります。 応答メッセージは通常の SOAP メッセージまたは SOAP 障害です。

- プロバイダー・モードで、CICS が In-Out MEP を使用する Web サービスから要求メッセージを受け取るとき、応答メッセージがリクエスターに戻されます。
- リクエスター・モードでは、CICS は要求メッセージを送信して、応答を待機します。 この応答は、正規応答メッセージまたは SOAP 障害メッセージのどちらかです。 CICS が応答を待機する時間の長さは、パイプラインで構成されて、そのパイプラインを使用するすべての Web サービスに適用されます。 CICS が応答を受け取る前に要求がタイムアウトになる場合、サービス・リクエスター・アプリケーションにエラーが戻されます。

In-Optional-Out

要求メッセージが Web サービス・プロバイダーに送信され、応答メッセージはオプションで Web サービス・リクエスターに戻ります。 応答がある場合、応答は通常の SOAP メッセージまたは SOAP 障害のいずれかです。

- プロバイダー・モードでは、SOAP 応答メッセージまたは SOAP 障害のどちらを戻すか、または何も戻さないかの判断は実行時に行われて、サービス・プロバイダーのアプリケーション・ロジックに依存しています。 CICS が応答を Web サービス・リクエスターに送信しない場合、DFHNORESPONSE コンテナーが SOAP ハンドラー・チャネルに入れられて、パイプラインが応答メッセージを送信してはならないことが示されます。 メッセージが送信されない場合、サービス・プロバイダー・アプリケーションは DFHWS-DATA コンテナーをチャネルから削除する必要があります。
- リクエスター・モードでは、CICS は要求メッセージを送信して、Web サービス・リクエスターからの応答を待機します。応答を受け取る前に要求がタイムアウトになる場合、CICS は、メッセージが正常に受け取られたためにプロバイダーは応答を送信する必要がなかったと想定します。 CICS が応答を待機する時間の長さは、パイプラインで構成されて、そのパイプラインを使用するすべての Web サービスに適用されます。

Robust In-Only

要求メッセージが Web サービス・プロバイダーに送信され、エラーが生じ

た場合にのみ、Web サービス・リクエスターに応答メッセージが戻ります。 エラーがある場合は、SOAP 障害メッセージがリクエスターに送信されます。

- プロバイダー・モードでは、パイプラインが要求メッセージをアプリケーションに正常に渡した場合、 DFHNORESPONSE コンテナーが SOAP ハンドラー・チャネルに入れられて、パイプラインが応答メッセージを送信してはならないことが示されます。 パイプラインでエラーが生じた場合は、SOAP 障害メッセージがリクエスターに戻されます。
- リクエスター・モードでは、CICS は要求メッセージを Web サービス・プロバイダーに送信して、指定された期間待機してからタイムアウトします。 CICS が応答を待機する時間の長さは、パイプラインで構成されて、そのパイプラインを使用するすべての Web サービスに適用されます。 タイムアウトがある場合、CICS は要求メッセージが正常に受け取られたと想定します。

WSDL 2.0 でのメッセージ交換パターンの詳細については、以下の W3C 仕様を参照してください。

- *WSDL 2.0 Part 2: Adjuncts*:。
- *WSDL 2.0 Part 2: Additional MEPs*: 。

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、その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・ファイルの情報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。

外部標準

Web サービスの CICS サポートは、多数の業界標準および仕様に準拠しています。

SOAP アーキテクチャーおよびメッセージ形式

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、さまざまな基礎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、World Wide Web Consortium (W3C) で開発され、W3C が発行した以下の文書で定義されています。 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、以下の情報も含まれています。

- 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 Web サービスのアーキテクチャー

SOAP 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。
  - 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。
  - 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・レジストリーは、サービス・プロバイダーがサービス記述を発行でき、サービス・リクエスターがこれらのサービス記述を検出できる中心的な場所です。

サービス・リクエスターとプロバイダーがこの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、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・コンポーネントです。 例えば、サービスを提供する組織はさまざまな方法でサービス記述をサービスのユーザーに直接配布することができます (例えば FTP サイトからのダウンロードとしてサービスを提供できます)。

サービス・レジストリーを使用することには、リクエスターとプロバイダーの両方にとってさまざまな利点があります。例えば、IBM® WebSphere® Service Registry and Repository (WSRR) を使用すると、リクエスターはサービスをより速く見つけることができ、プロバイダーは提供されるサービスのバージョン管理を容易に行うことができます。

CICS は、サービス・リクエスター・コンポーネントとサービス・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。ただし、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。 IBM WebSphere Service Registry and Repository (WSRR) を使用する場合、CICS は Web サービス・アシスタントを介して WSRR をサポートします。 または、別のプラットフォームにサービス・レジストリーを配置することもできます。

サービス・プロバイダー、サービス・リクエスター、およびサービス・レジストリーの間の対話

サービス・プロバイダー、サービス・リクエスター、およびサービス・レジストリーの間の対話では、次のような動作が行われます。

Publish

サービス・レジストリーを使用する際、サービス・プロバイダーは自分のサービス記述をサービス・レジストリーに発行 (publish) して、サービス・リクエスターがそれを見つけられるようにします。

Find サービス・レジストリーを使用する際、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。

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。

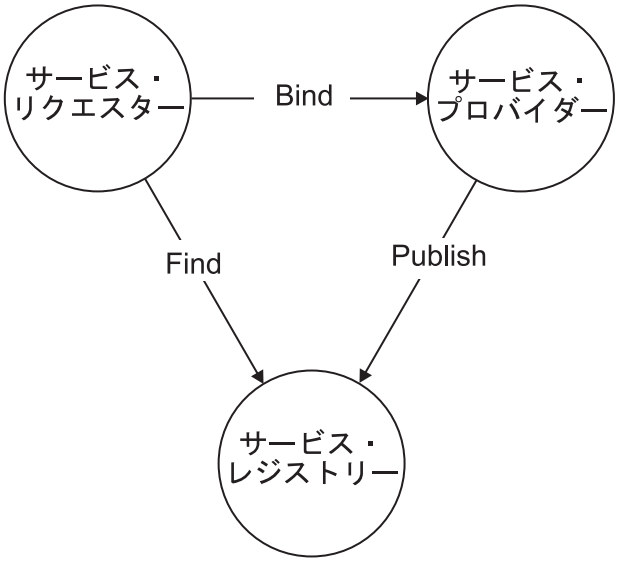


図 *8. Web* サービスのコンポーネントおよび対話

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 サービスを開始する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用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。

サービス記述は WSDL 1.1 または WSDL 2.0 のいずれかの仕様に適合できます。それぞれにおいて、サービス記述に含めることができる用語と主要なエレメントが異なります。 以下の情報は、サービス記述の目的を説明するために、WSDL 1.1 の用語とエレメントを使用します。

WSDL の構造により、サービス記述は次のように 2 つの定義に区分されます。

- 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、サービスの実装と開始を行うプログラムを作成できるようにします。
- 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、およびその実装に固有のその他の詳細を記述します。 これにより、サービス・リクエスターがサービス・プロバイダーに接続できます。

21 ページの図 9を参照してください。

WSDL 1.1 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・タイプから構成される場合は、<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。

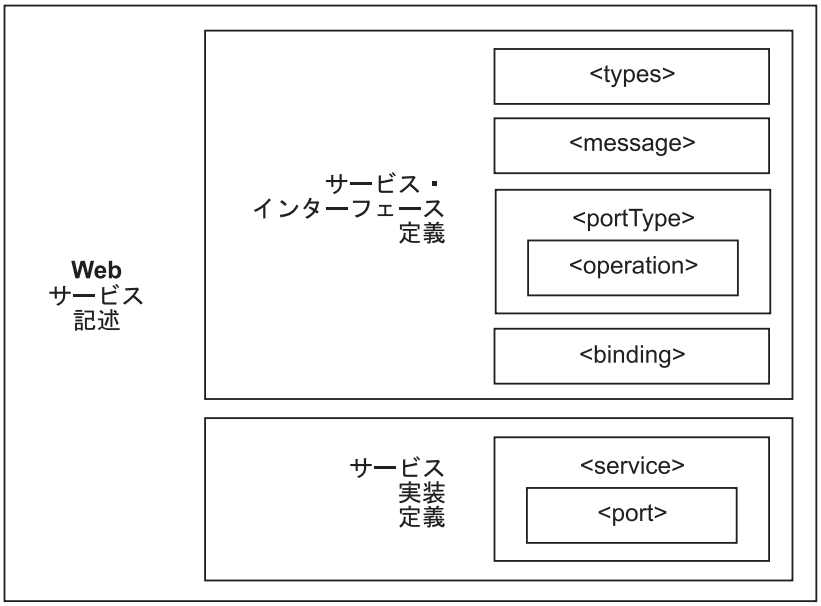


図 *9. Web* サービス記述の構造

Web サービス記述を区分できるため、サービス記述全体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**:**

いくつかの異なるメカニズムを使用して、サービス記述を発行できます。 それぞれのメカニズムは、さまざまな状況での使用に適合したものになっています。 CICS では、IBM WebSphere Service Registry and Repository (WSRR) を使用してサービス記述を発行することが可能です。 または、他の方式を使用してサービス記述を発行することもできます。

WSSR

CICS では WSRR を使用してサービス記述を発行することができます。

CICS での WSSR サポートの詳細については、インフォメーション・センターのトピック『Web サービス・アシスタントと WSRR の相互運用性』を参照してください。

以下の機構はいずれも CICS では直接サポートされていませんが、サービス記述を発行するために CICS で使用することが可能です。

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 E メール添付ファイル、FTP サイト、または CD ROM 配布を使用して、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。

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 はまた、企業が自社で提供する Web サービスに関する情報を登録して他の企業から検索できるようにする仕様の、一般にアクセス可能な一連の実装でもあります。

必要に応じて、複数の形式でサービス記述を発行することができます。

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> 内に格納されている <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・エレメントです。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入ります。

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。

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す。ここにはメッセージの最終の受信側が利用するための情報が格納されます。

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 10 には、SOAP メッセージの主要なエレメントを示します。

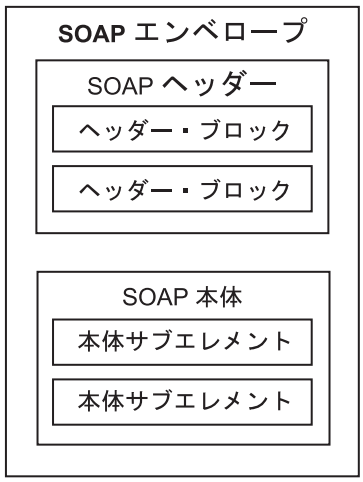


図 *10. SOAP* メッセージの構造

24 ページの図 11 は、ヘッダー・ブロック (`<m:reservation>` エレメントと `<n:passenger>` エレメント) および本体 (`<p:itinerary>` エレメントと `<q:lodging>` エレメントを含む) を格納する SOAP メッセージの例です。

```
<?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 *11. 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 エレメントです。これは、送信側から最後の受信側までのメッセージ経路の中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。 しかし、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。このようにして、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。 以下の重要な属性に注目してください。

encodingStyle
SOAP メッセージの一部をエンコードするために使用する規則を示します。 SOAP では、XML が許可する非常に柔軟なエンコード方式よりも限られた、一連のデータのエンコード規則を定義します。

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** 属性に同じ機能があります。

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、以下のとおりです。

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 障害:

SOAP <Fault> エレメントには、SOAP メッセージ内のエラーおよび状況情報が格納されます。

Web サービスでエラーが発生した場合、障害メッセージがクライアントに戻されます。 障害メッセージの基本構造は、SOAP 仕様で定義されています。 それぞれの障害メッセージには、特定のエラー状態を記述する XML を含めることができます。 例えば、CICS Web サービスでアプリケーションの異常終了が発生した場合、その異常終了を報告する障害メッセージがクライアントに戻されます。

CICS では、以下のさまざまなタイプの障害メッセージを送信できます。

- 標準の SOAP 障害メッセージ。これは、SOAP 仕様または CICS でサポートされている Web サービス仕様で定義されます。この障害は、一般的なエラー条件 (SOAP エンベロープの形式が正しくないなど) を報告します。
- アプリケーション SOAP 障害メッセージ。これは、アプリケーションによって検出または処理される状態に対する応答として、**EXEC CICS SOAPFAULT** API コマ

ンドを使用して生成されます。 これらの障害メッセージの構造は、アプリケーションでは認識されますが、CICS では認識されません。

- SOAP ハンドラー障害メッセージ。これは、CICS での一般的なエラー処理に対する応答として、SOAP ハンドラー・プログラムによって生成されます。 例えば、SOAP ハンドラー・プログラムは、異常終了の SOAP 障害、XML 解析の障害、または他の一般的なエラーなどを送信します。
- アプリケーション・ハンドラー障害メッセージ。これは、SOAP メッセージの本文の処理時のエラーの検出に対する応答として、CICS SOAP アプリケーション・ハンドラーによって生成されます。 これらの障害は、バイナリー・アプリケーション・データへの XML の変換処理中、または応答の生成時に発生します。

エラーが発生した場合、SOAP <Fault> エレメントが本文の項目になる必要があります。これは、<Body> エレメント内に複数存在することはできません。 SOAP エレメント <Fault> の中に置かれる XML 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、障害を生成した 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> エレメントの内容が正常に処理されなかった場合です。 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、ヘッダー項目に格納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> エレメント内の必須エレメントの 1 つです。 <Reason> エレメントには、障害に関する情報をさまざまなネイティブ言語で記述した <Text> エレメントを 1 つ以上組み込みます。

<Node>
<Node> エレメントには、障害を生成した 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、障害が発生した時点でノードが持っ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 障害の例およびスキーマ

以下の例では、SOAP メッセージの本文の処理時に、アプリケーション・ハンドラー DFHPITP によって生成される SOAP 障害メッセージが示されています。

```
<SOAP-ENV:Fault xmlns="">
   <faultcode>SOAP-ENV:Server</faultcode>
   <faultstring>Conversion to SOAP failed</faultstring>
      <detail>
         <CICSFault xmlns="http://www.ibm.com/software/htp/cics/WSFault">
          DFHPI1008 25/01/2010 14:16:50 IYCWZCFU 00340 XML
          generation failed because of incorrect input
          (CONTAINER_NOT_FOUND container name) for WEBSERVICE
          servicename.
          </CICSFault>
      </detail>
</SOAP-ENV:Fault>
```

この例の内容のほとんどは、すべての障害メッセージに共通のものです。 <Detail> エレメントには、アプリケーション・ハンドラーによって検出された問題を記述する固有の情報が含まれています。 この固有の障害メッセージには、CICS メッセー

ジ・ログに記録されているエラー・メッセージのコピーが含まれています。 アプリケーション・ハンドラーの SOAP 障害をプログラマチックに解析する場合、以下の XML スキーマを使用します。

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.ibm.com/software/htp/cics/WSFault"
xmlns:tns="http://www.ibm.com/software/htp/cics/WSFault"
elementFormDefault="qualified">
   <element name="CICSFault" type="string">
      <annotation>
         <documentation>
            The value of this element is a text string that describes a
            problem encountered during the processing of the Body of a
            SOAP message.
         </documentation>
      </annotation>
   </element>
</schema>
```

<Detail> セクションはさまざまな方法で構成できるため、汎用の障害メッセージはこれより複雑になります。 SOAP ハンドラーの障害をプログラマチックに解析する場合、*usshome*/schemas/soapfault/soapfault.xsd (*usshome* は **USSHOME** システム初期設定パラメーターの値) で提供されている XML スキーマを使用します。

SOAP Web サービス使用の計画立案

CICS で SOAP 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。

始める前に

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、ラッパーを簡単に作成できる場合があります。

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、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。

SOAP を使用しないことにした場合は、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。

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。

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、該当する言語にサポートされていないデータ・タイプと構造のリストをチェックし、アプリケーション・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。アシスタントがサポートできるものとサポートできないものについて詳しくは、高水準言語と XML のスキーマ・マッピングを参照してください。

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、 IBM Developer for Z などのツールを使用して必要な成果物を作成し、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ための独自のコードを提供することができます。また、提供されるベンダーのインターフェース API を使用することもできます。

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。 以下に例を示します。

- 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。

次のタスク

CICS および JSON Web サービス

CICS で JSON Web サービスを始めるには、いくつかの方法があります。最適な方法は、Web サービスの使用に関する習得済みの知識の量や計画の進捗度によって異なります。

このタスクについて

CICS では、JSON ベースのサービスとしてリソースを公開するための複数の異なるテクノロジーをサポートしています。このセクションは、「JSON Web サービス」と呼ばれる、より古いテクノロジーに関連しています。z/OS® Connect for CICS 1.0 に関する同等の情報については、 40 ページの『CICS と z/OS Connect』を参照してください。

JSON Web サービスは、CICS プログラムを JSON サービスとして使用可能にするためのテクノロジーです。これらは RESTful サービスであるか、または要求/応答のリモート・プロシージャー・コール・スタイルのサービスである場合があります。このテクノロジーは、SOAP Web サービスに使用されるものから派生し、WSBind ファイルの生成に使用される DFHLS2JS および DFHJS2LS という JCL プロシージャーを使用します。これらの WSBind ファイルは、WEBSERVICE リソースとして CICS にデプロイされます。WEBSERVICE リソースでは、JSON とアプリケーション・バイナリー・データ・フォーマットの間の自動変換を支援します。

JSON 用の IBM の主要なテクノロジーには、z/OS Connect Enterprise Edition という製品が含まれています。この製品は、CICS JSON Web サービスと広く互換性がありますが、多くの追加の統合オプションおよび機能を提供します。

このオプションの詳しい情報については、z/OS Connect 入門を参照ください。
CICS での JSON 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。

手順

- アプリケーションの例をインストールします。 CICS には、JSON 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。これを行うには、DFHLS2JS を使用して、提供される言語構造から Web サービスを生成します。 Web ブラウザーまたはサード・パーティー・クライアント・アプリケーションを使って JSON Web サービスをテストすることもできます。詳細については、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を参照してください。

  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。実例アプリケーションについては、CICS カタログ・マネージャーの実例アプリケーションで説明します。
- サービス・プロバイダーとしてアプリケーションを配置する計画を行います。 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計画を開始する方法については、既に十分な知識があることが前提となっています。

- JSON Web サービスの別のオプションには、z/OS Connect の使用があります。このオプションの詳しい情報については、z/OS Connect 入門を参照ください。

JSON の Web サービスの概念

JSON の Web サービスの背景にある概念について理解するには、このトピックをお読みください。

Web サービス

「Web サービス」は、ネットワーク上のアドレス可能なロケーションでホストされるソフトウェア機能を指す一般用語です。このような一般的な意味では、クラウド・ベースのサービスやユーティリティー・サービス、さらに部門アプリケーションを指す場合もあります。「Web サービス」という語は、さらに特定的な意味で使用されることもあります。例えば、WSDL 文書で記述された SOAP を使用してホストされるサービスなどがこれに該当します。通常、「CICS の Web サービス」という語が指すのは、こちらのより特定的な意味の方です。しかし、JSON コミュニティーが JSON ベースのサービスに言及するときには、多くの場合、より一般的な用語を使用します。JSON Web サービスではこの用語が一般的な意味で使用されています。

SOAP と JSON には、以下のような重要な相違点があります。

- SOAP メッセージの内容は XML データであるのに対し、JSON メッセージの内容は JSON データです。JSON と XML では、構造化データを記述するためのエンコード・メカニズムが異なります。JSON のエンコード・メカニズムの方がより効率的である傾向にあるため、一般的な JSON メッセージは同等の XML メッセージよりサイズが小さくなります。
- JSON は簡単に JavaScript アプリケーションに統合できますが、XML はそうではありません。そのため、多くのモバイル・アプリケーション開発者は、JSON のデータ形式を好んで使用します。
- SOAP には、メッセージにヘッダーを追加するメカニズムがあり、サービス品質の仕様ファミリー (セキュリティー構成、分散トランザクションなど) があります。JSON にはこのメカニズムがない代わりに、基礎となる HTTP ネットワーク・プロトコルのサービスに依存しています。その結果、ワークロードを保護したり構成したりするためのオプションが少なくなっています。
- SOAP の Web サービスは、WSDL 文書を使用して記述されます。JSON の Web サービスの構造はあまり形式的ではありません。疎結合の傾向にあり、例示ドキュメンテーションが主流です。
- SOAP の Web サービスには明示的なエラー・フォーマットがあり、SOAP 障害メッセージもこれに含まれます。JSON には、これに相当するものはありません。

JSON と SOAP には、以下のように類似点も多数あります。

- JSON の CICS 実装は、SOAP アーキテクチャーから派生し、概念や成果物の多くを共用します。

- どちらにもオフライン・ユーティリティー・プログラムがあり、アプリケーション・データと外部データ表現の相互マッピングを支援します。SOAP には DFHLS2WS および DFHWS2LS があり、JSON には DFHLS2JS および DFHJS2LS があります。
- どちらのテクノロジーのデプロイメント・メカニズムでも、PIPELINE リソース、WEBSERVICE リソース、および URIMAP リソースが使用されます。

JSON スキーマ

SOAP と比べて JSON が不利な点として、JSON インターフェースの構造を文書化することの難しさがあります。 SOAP Web サービスには WSDL 文書および XML スキーマという利点があります。WSDL 文書を理解することは容易ではありませんが、WSDL 文書とともに使用できるツールが多数あります。

JSON の場合、これに最も近いのが JSON スキーマ仕様で、http://json-schema.org/ から入手できます。現時点では、これはドラフト仕様ですが、IETF 標準化プロセスに進む過程にあります。CICS JSON アシスタント (DFHLS2JS および DFHJS2LS) は、この最新仕様のドラフト 4 を部分的に実装しています。DFHLS2JS を使用すると、JSON スキーマを生成でき、DFHJS2LS を使用するとそれを処理できます。

JSON スキーマを使用すると、CICS で実装された JSON Web サービスの有効な構文およびコンテンツ・モデルについて理解できます。JSON スキーマ仕様には、XML スキーマ仕様と同じツール・エコシステムはありませんが、このデータ形式を使用するための新世代の JSON ツールが作成される可能性があります。

JSON ベースの Web サービスの CICS 実装

CICS は、3 つのモード (z/OS Connect モード、要求/応答モードと RESTful モード) の JSON Web サービスをサポートします。また、CICS は、アプリケーション自身で JSON データと COBOL スタイルのデータ形式の間の相互変換を実行できるというプログラマチックなシナリオもサポートします。

z/OS Connect

>z/OS Connect を使用して、CICS プログラムを JavaScript Object Notation (JSON) インターフェースで呼び出すことができます。z/OS Connect for CICS は当初、CICS TS Feature Pack for Mobile Extensions で提供されていた Java パイプラインの JSON 機能に代わるものとして導入され、CICS TS バージョン 5.2 に統合されました。その後、z/OS Connect には機能が追加され、別個の製品として発展し、z/OS Connect Enterprise Edition と呼ばれるようになりました。

z/OS Connect は、CICS で JSON サービスおよび API を実装するための IBM の主要なテクノロジーです。これは、3 つのバージョンで使用可能で、いくつかのデプロイメント・オプションをサポートします。z/OS Connect for CICS は、エントリー・レベルのエディションで、低コストのオプションとなりますが、他のバージョンの機能拡張は含まれていません。z/OS Connect EE バージョン 2.0 および 3.0 は、さらに広い範囲の統合オプションを提供します。

z/OS Connect EE は、個別に注文可能な IBM 製品です。これは、JSON サービスのサポートが含まれていた z/OS Connect for CICS の機能に基づいて構築されています。z/OS Connect EE を使用して、API 開発者は JSON サービスから JSON API を構成できます。API は、z/OS Connect EE に付属する Eclipse ベースの API エディターで構成およびパッケージ化した後、z/OS Connect ランタイムにデプロイされます。API パッケージには Swagger 2.0 定義が含まれており、開発者はこれを使用して API をアプリケーションに簡単に組み込むことができます。サービス呼び出しに関する権限セキュリティー検査、システム管理機能 (SMF) レコードの作成、および RESTful サービス要求のロギングなどの主な z/OS Connect 機能も API に適用されます。

要求/応答

要求/応答 JSON パターンは、CICS の SOAP ベース Web サービスのパターンと非常によく似ています。この Web サービスは、CICS で PROGRAM を使用して実装されます。PROGRAM の入出力データ形式は、言語構造 (COBOL コピーブックなど) を使用して記述します。着信 JSON メッセージからアプリケーション・データへの変換、およびアプリケーションへのリンクは、CICS が行います。アプリケーションは出力データを CICS に返し、CICS はそのデータを JSON データに変換して、クライアントに返します。

このシナリオでは、JSON クライアントは、HTTP POST メソッドを使用して CICS に接続する必要があります。

要求/応答モードの JSON Web サービスは、ボトムアップ・モードとトップダウン・モードのいずれでも開発できます。ボトムアップ・モードでは、既存の CICS PROGRAM が、DFHLS2JS JSON アシスタントを使用して JSON Web サービスとして公開されます。トップダウン・モードでは、既存の JSON スキーマを使用して記述されたインターフェースを実装するための、新規 JSON Web サービスを開発できます。トップダウン・モードでは、DFHJS2LS JSON アシスタントを使用して新規言語構造が生成されます。これらの構造を使用するために、アプリケーションを作成する必要があります。

要求/応答パターンを使用して、COMMAREA またはチャネル接続された CICS PROGRAM をターゲットとする JSON Web サービスを作成できます。要求/応答 JSON Web サービスは、プロバイダー・モードでのみ使用できます (この場合、CICS はサーバーとして機能します)。

RESTful

このシナリオは、SOAP Web サービスのシナリオとは異なります。RESTful JSON Web サービスの概念については、RESTful JSON Web サービスの概念でより詳しく説明しています。 RESTful JSON Web サービスは、REpresentational State Transfer (REST) 設計パターンのアーキテクチャー規則を実装します。この設計パターンは、既存の CICS アプリケーションとはほぼ無関係なため、トップダウン・モードでのみ使用できます。

JSON スキーマを処理するには、RESTful モードで DFHJS2LS を使用します。サービスを実装するために、アプリケーションを作成する必要がありま

す。このアプリケーションは、着信要求に使用された HTTP メソッドに応じて異なる動作をする必要があります。

CICS は、純粋なスタイルの RESTful アプリケーションを実装します。このようなアプリケーションでは、POST (作成)、GET (照会)、および PUT (置換) のデータ形式が同じです。

RESTful JSON Web サービス・アプリケーションは、チャネル・ベースのプログラム・インターフェースを使用する必要があります。COMMAREA はサポートされていません。RESTful JSON Web サービスは、プロバイダー・モードでのみ使用できます (この場合、CICS はサーバーとして機能します)。

プログラマチック・モード

このシナリオでは、アプリケーションは CICS 提供プログラム DFHJSON に LINK でき、アプリケーション・データと JSON データの相互変換をこのプログラムに依頼できます。例えば、アプリケーションは、リモート JSON Web サービスに送信する JSON データを生成するために、この機能を使用できます。この場合、アプリケーションは、CICS WEB API を使用して、リモート JSON Web サービスに接続する必要があります。

CICS には、リクエスター・モードの JSON Web サービスに対する組み込みサポートはありませんが、アプリケーションは、プログラマチック・モードを活用することによって、リモート JSON Web サービスを呼び出すことができます。

RESTful JSON Web サービスの概念

RESTful の Web サービスの背景にある概念について理解するには、このトピックをお読みください。

RESTful Web サービス

REpresentational State Transfer、つまり REST は、サーバーに保管されているリソースと対話するための設計パターンです。各リソースには ID、データ・タイプがあり、一連のアクションをサポートしています。

RESTful 設計パターンは通常、インターネットの言語である HTTP と組み合わせて使用します。このコンテキストにおいて、リソースの ID とはその URI であり、データ・タイプとはそのメディア・タイプであり、アクションは標準の HTTP メソッド (GET、 PUT、POST、および DELETE) で構成されています。

このスタイルのサービスは、要求/応答スタイルの Web サービスとは異なります。

- 要求/応答サービスはアプリケーションとの対話を開始するのに対し、RESTful サービスは一般的にデータ (「リソース」と呼ぶ) と対話します。
- 要求/応答サービスはアプリケーションで定義されている「操作」を行いますが、RESTful サービスはアプリケーション特定の概念を回避します。
- 要求/応答サービスではメッセージごとに異なるデータ形式が使用されますが、RESTful サービスでは一般的に各種 HTTP メソッド間でデータ形式が共用されます。

4 つの主要な HTTP メソッドにより、RESTful サービスで一般的に実装される 4 つの操作が定義されます。HTTP POST メソッドはリソースを作成するために使用され、GET はその照会に、PUT は変更に、DELETE は破棄に使用されます。最も一般的に使用される RESTful アーキテクチャーには、これらの 4 つの操作で使用される共用データ・モデルがあります。このデータ・モデルでは、POST メソッド (作成) への入力、GET メソッド (照会) の出力、PUT メソッド (置換) への入力を定義します。この単純な設計パターンは、RESTful コミュニティー内で頻繁に使用されていますが、これが唯一の RESTful 設計パターンというわけではありません。操作の成功または失敗を示すために、HTTP 状況コードを使用します。 RESTful API の中には、他の方法で設計されているものもあります。

RESTful Web サービスでは、「HEAD」という名前の 5 番目の HTTP メソッドがサポートされる場合があります。このメソッドは GET に相当します。ただし、HTTP ヘッダーだけを返して、本体データは返しません。これは、リソースの存在をテストするために使用されることがあります。この HEAD メソッドは、すべての RESTful API でサポートされているわけではありません。

従来の CICS アプリケーションが RESTful アーキテクチャー・パターンと一致することはほとんどありません。一般的な CICS アプリケーションは、複数の操作を実装し、その各操作に入出力形式のデータ・モデルがあります。これらの既存の操作がこの 4 つの HTTP メソッドに直接対応する可能性はほとんどありません。このため、RESTful アーキテクチャー・パターンは主に、CICS の新しいアプリケーションを対象としています。既存の CICS アプリケーションを RESTful サービスとして公開するには、RESTful 規則に準拠する新しいインターフェースでアプリケーションをラップすることが必要になる可能性があります。

URI

RESTful サービスの ID は、URI によって示されます。URI は、ホスト名、ポート番号、パス、およびオプションとして照会ストリングなど、幾つかの構成要素からなっています。ターゲットとして CICS の TCPIPSERVICE リソースを指定するために、ドメイン名とポート番号を一緒に指定します。詳細については、TCPIPSERVICE リソースを参照してください。 URI パスは修飾子であり、これだけでサービスを一意的に識別できる場合があります。ただし、多くの RESTful Web サービスでは、正確なリソースを識別するために追加の照会ストリングが使用されます。以下の例について考慮します。

- `http://www.example.org:10000/JSONServices/AccountService`
- `https://www.example.org:10000/JSONServices?Service=Account`

最初の例の場合、URI パスは `JSONServices/AccountService` です。2 番目の例のパスは `JSONServices` であり、`Service=Account` という追加の照会ストリングがあります。どちらのスタイルの URI も、JSON では受け入れ可能と見なされます。これは、SOAP と比較して重要な相違点です。SOAP では、最初のスタイルの方が望ましい URI です。

CICS JSON サービスは、z/OS Connect Enterprise Edition を使用して RESTful サービスに変換できます。URI フラグメントと HTTP ヘッダーを既存のコピーブックのフィールドにマップするためにグラフィカル・ユーザー・インターフェースが使用され、さまざまなプログラムが各 HTTP メソッドのターゲットとして機能す

る可能性があります。既存のアプリケーション・アセットから RESTful サービスを構成する機能は、CICS における他の JSON テクノロジーに対する z/OS Connect の主な利点の 1 つです。

CICS には、限定された形式の RESTful サービスを実装するための、より古いテクノロジーも備えられています。URIMAP リソースを使用して、インバウンド・メッセージの処理時に使用する適切な WEBSERVICE および PIPELINE を識別できます。URIMAP では、特定の PIPELINE や WEBSERVICE に URI をマップすることができ、そのマッピングには、その URI の照会ストリングが含まれる可能性があります。

CICS では URIMAP リソースを使用して、インバウンド・メッセージを処理する際に使用する適切な WEBSERVICE と PIPELINE を識別します。URIMAP では、`path` 属性の一部として照会ストリングを使用できます。そのため、URIMAP はどちらのタイプの URI の使用にも適しています。

JSON Web サービス使用の計画立案

CICS で JSON 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。

始める前に

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、ラッパーを簡単に作成できる場合があります。

データ構造と **JSON** スキーマ間のマッピングを生成するために **CICS** アシスタントを使用する予定ですか**?**

> JSON 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、JSON 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 JSON メッセージにマップする既存のコードがある場合は、JSON 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。
>
> CICS 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、該当する言語にサポートされていないデー

タ・タイプと構造のリストをチェックし、アプリケーション・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。 アシスタントがサポートできるものとサポートできないものについて詳しくは、高水準言語と JSON のスキーマ・マッピングを参照してください。

JSON 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・プロバイダーで簡単に配置できます。 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 12 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。

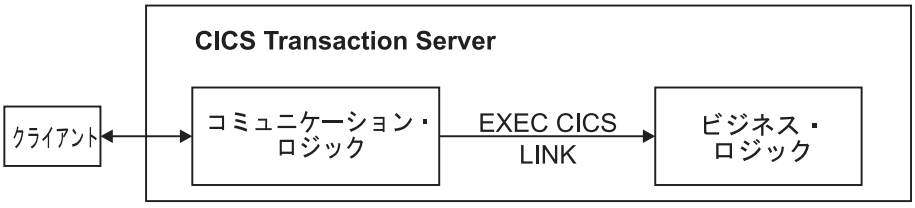


図 *12.* コミュニケーション・ロジックとビジネス・ロジックに分割されたアプリケーション

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 13 に図示されています。

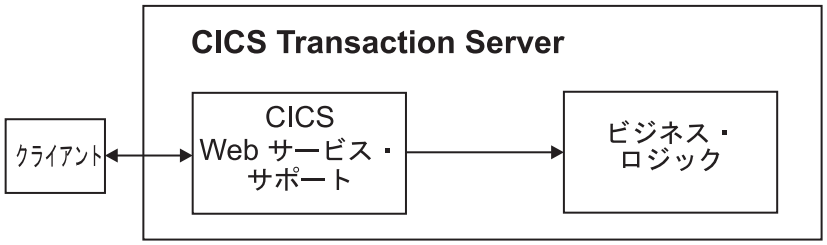


図 *13. Web* サービス・プロバイダーとしての *CICS* アプリケーションの単純な配置

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。

z/OS Connect を使用して **JSON** スキーマまたは **Swagger** 文書とアプリケーション・データ構造の間のマッピングを生成する場合**:**

z/OS Connect Enterprise Edition は、CICS アシスタントがサポートするよりも広い範囲のアプリケーション・コピーブック構造をサポートします。 ほとんどの場合、これらのコピーブックは未変更のままサポートされます。 そうでない場合は、z/OS Connect サービスとビジネス・ロジックの間にラッパー・プログラムを介在させることができます。

CICS アシスタントを使用して **JSON** スキーマ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 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 Web サービス記述に適合するサービスを提供するために既存のプログラムを配置する場合は、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 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 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 アシスタントが JSON 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。

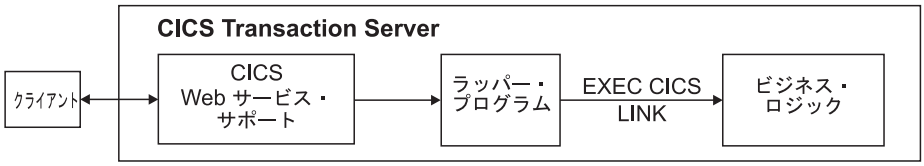


図 *14.* ラッパー・プログラムを使用した、*Web* サービス・プロバイダーとしての *CICS* アプリケーションの配置

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

エラー処理

CICS アシスタントを使用する予定がある場合は、エラーが発生したときに変更のロールバックを処理する方法についても検討する必要があります。サービス・リクエスターから JSON 要求メッセージを受信すると、JSON メッセージはアプリケーション・プログラムに渡される直前に CICS によって変換されます。この変換中にエラーが発生すると、CICS はメッセージで実行された作業を自動的にロールバックしません。例えば、パイプラインでハンドラーを使用して JSON メッセージに別の処理を追加する予定がある場合、既に実行したリカバリー可能な変更をロールバックするかどうかを決定する必要があります。

アウトバウンド JSON メッセージでは、サービス・プロバイダー・アプリケーション・プログラムがサービス・リクエスターに応答メッセージを送信するような場合は、応答 JSON メッセージの生成時に CICS がエラーを検出すると、アプリケーション・プログラムによって行われたリカバリー可能な変更はすべて自動的にバックアウトされます。ご使用のアプリケーション・プログラムにとって同期点を追加することが適切かどうかを検討する必要があります。

JSON サービス・リクエスター・アプリケーションの計画

CICS は、リクエスター・モードの JSON Web サービスの組み込みサポートを提供していません。これらのサービスは、z/OS Connect Enterprise Edition V3.0 によって提供されます。

z/OS Connect について詳しくは、『CICS と z/OS Connect』を参照してください。z/OS Connect を使用せずに CICS アプリケーションからリモート JSON Web サービスを呼び出すには、**EXEC CICS WEB** API コマンドを使用します。詳細については、JSON Web サービス・アプリケーションの作成を参照してください。

CICS と z/OS Connect

z/OS Connect を使用して、CICS プログラムを JavaScript Object Notation (JSON) インターフェースで呼び出すことができます。

z/OS Connect for CICS 1.0 は当初、CICS TS Feature Pack for Mobile Extensions で提供されていた Java Pipelines for JSON の JSON 機能に代わるものとして導入され、CICS TS バージョン 5.2 に統合されました。その後、z/OS Connect には機能が追加され、別個の製品として発展し、z/OS Connect Enterprise Edition と呼ばれるようになりました。ここでは、代替オファリングの相違点と、実装の手順について説明します。

z/OS Connect は、CICS において JSON サービスおよび API を実装するための IBM 最高のテクノロジーです。これは、3 つのバージョンで使用可能で、いくつかのデプロイメント・オプションをサポートします。z/OS Connect for CICS 1.0 は、エントリー・レベル・エディションで、低コストのオプションとなりますが、他のバージョンの機能拡張は含まれていません。z/OS Connect Enterprise Edition バージョン 2.0 および 3.0 は、さらに広い範囲の統合オプションを提供します。

JSON サービスの概要については、JSON 要求のサービス・プロバイダーとしてのCICSを参照してください。

z/OS Connect Enterprise Edition に関するメンテナンスの考慮事項

z/OS Connect Enterprise Edition は、独立して実行することも、CICS に組み込まれた Liberty で実行することもできるため、2 つの環境を一意的に記述する必要があります。両方の環境は並行して実行できますが、相互に排他的ではないため、BBOA* モジュールを含む WebSphere Liberty Profile (WLP) クライアント・ライブラリーに保守を適用するときは注意する必要があります。

詳しくは、CICS TS 5.3 と z/OS Connect EE 2.0 の保守の同期を保持するを参照してください。

z/OS Connect Enterprise Edition

z/OS Connect Enterprise Edition は、個別に注文可能な IBM 製品です。これは、JSON サービスのサポートが含まれていた z/OS Connect for CICS 1.0 の機能に基づいて構築されています。z/OS Connect Enterprise Edition を使用して、API 開発者は JSON サービスから JSON API を構成できます。API は、z/OS Connect Enterprise Edition に付属する Eclipse ベースの API エディターで構成およびパッケージ化した後、z/OS Connect ランタイムにデプロイされます。API パッケージには Swagger 2.0 定義が含まれており、開発者はこれを使用して API をアプリケーションに簡単に組み込むことができます。サービス呼び出しに関する権限セキュリティー検査、システム管理機能 (SMF) レコードの作成、および RESTful サービス要求のロギングなどの主な z/OS Connect 機能も API に適用されます。

z/OS Connect Enterprise Edition は、CICS 内の Liberty JVM サーバーで実行されるように構成できます。CICS の Liberty JVM サーバーで実行される z/OS Connect Enterprise Edition サーバーは、サービス・プロバイダーを使用して CICS アプリケーション・プログラムに接続します。z/OS Connect Enterprise Edition V3.0 では、構成に使用する CICS サービス・プロバイダーを選択できます。

- z/OS Connect Enterprise Edition V2.0 を使用している場合は、CICS TS および機能 `cicsts:zosConnect-2.0` で提供される CICS サービス・プロバイダーを使用します。この構成により、CICS プログラムへのローカル・ハイパフォーマンス接続が使用可能になります。これは、z/OS Connect Enterprise Edition V2.0 でサポートされる唯一のサービス・プロバイダーです。
- z/OS Connect Enterprise Edition V3.0 を使用してローカル CICS 領域に接続する場合は、z/OS Connect Enterprise Edition V3.0 および機能 `zosconnect:cicsService-1.0` で提供される CICS サービス・プロバイダーを使用します。この構成では、ローカル CICS 領域にローカル最適化接続が使用されます。
- z/OS Connect Enterprise Edition V3.0 を使用してリモート CICS 領域に接続する場合は、z/OS Connect Enterprise Edition V3.0 および機能 `zosconnect:cicsService-1.0` で提供される CICS サービス・プロバイダーを使用します。この構成では、リモート CICS 領域に IP 相互接続 (IPIC) 接続が使用されます。

z/OS Connect for CICS 1.0 で使用するように作成されていた JSON サービスや、Java Pipelines for JSON で使用するように作成されていた大部分の JSON サービスは、z/OS Connect Enterprise Edition にホストできます。詳細については、 42 ページの『z/OS Connect for CICS 1.0』を参照してください。ただし、z/OS Connect Enterprise Edition に付属する API エディターのユーザーは、いくつかの制約事項を把握しておく必要があります。 詳しくは、z/OS Connect Enterprise Edition からの API の使用を参照してください。

詳しくは、z/OS Connect Enterprise Edition V3.0 の製品資料を参照してください。

z/OS Connect for CICS 1.0

z/OS Connect for CICS 1.0 は、CICS TS のバージョン 5.2 から提供されている無償の機能です。これは、概して Java Pipelines for JSON と同等のものであり、大部分の JSON Web サービスは、アプリケーションや WSBind ファイルを変更することなく、一方の環境から他方の環境に再デプロイできます。ただし、URI およびセキュリティー構成は、環境ごとに別にすることができます。z/OS Connect for CICS 1.0 にデプロイされる JSON Web サービスは、z/OS Connect Enterprise Edition にもデプロイできます。

CICS TS に組み込まれているバージョンの z/OS Connect は、WebSphere Liberty for z/OS の z/OS Connect の機能の大部分を共用しますが、CICS へのローカル・アクセスに関して最適化されており、標準の CICS サービスを使用します。WebSphere Liberty for z/OS の z/OS Connect に習熟している場合、これと z/OS Connect for CICS 1.0 の主な相違点は以下のとおりです。

- z/OS Connect for CICS 1.0 は、CICS の JSON Web サービスのデプロイメントのみをサポートしています。IMS やバッチなど、他の z/OS サブシステムをターゲットにして使用することはできません。
- z/OS Connect for CICS 1.0 は、CICS リソースと統合されます。JSON Web サービスは WEBSERVICE および URIMAP リソースを使用してデプロイされます。
- Java Pipelines for JSON からの JSON サービス有効化手法は、z/OS Connect では通常は使用できないものも含め、z/OS Connect for CICS 1.0 でサポートされています。
- JSON サービスは、ボトムアップ開発とトップダウン開発のどちらの方法でも開発できます。
- Java Pipelines for JSON 用に生成された JSON Web サービスを、z/OS Connect for CICS 1.0 に再デプロイできます。

WebSphere Liberty for z/OS の z/OS Connect について詳しくは、IBM z/OS Connect の概要を参照してください。

以前のテクノロジー

z/OS Connect より前から、その他の CICS のための JSON テクノロジーはありました (「Feature Pack for Mobile Extensions」や「JSON サービス」など)。Feature Pack for Mobile Extensions V1.0 は、CICS TS for z/OS V4.2 および V5.1 で適用されていました。この Feature Pack では、CICS アプリケーションを JSON ペイロードで RESTful Web サービスとして公開する機能、既存の JSON アプリケーションを呼び出す機能、および JSON とアプリケーション・データの間で双方向に変換する機能が提供されていました。CICS TS V5.2 以降、この機能は CICS TS に統合されました。古いテクノロジーでデプロイされたサービスは、通常、z/OS Connect と互換性があります。ただし、z/OS Connect Enterprise Edition によって、新しいサービスおよび API の最良のエクスペリエンスが提供されます。

z/OS Connect Enterprise Edition のデプロイメント・オプション

z/OS Connect Enterprise Edition には、3 つの主なデプロイメント・オプションがあります。

スタンドアロン **z/OS Connect Enterprise Edition**
: CICS への接続は、外部アドレス・スペースからの IPIC 接続または WOLA 接続を使用して行われます。詳しくは、z/OS Connect Enterprise Edition V3.0 製品資料内の『構成』を参照してください。

z/OS Connect 管理のサービスおよび **API** をホスティングする **CICS JVMSERVER** の **z/OS Connect Enterprise Edition V3**
: z/OS Connect Enterprise Edition V3.0 は、CICS 内の Liberty JVM サーバー内でホストされ、通信は、ローカルに最適化された IPIC 接続を介して行われます。サービスは、リッチな UI 主導型ツールを使用して作成され、SAR ファイルおよび AAR ファイルを使用してデプロイされます。この構成オプションは、スタンドアロン z/OS Connect Enterprise Edition オプションと類似していますが、CICS アドレス・スペースでホストされる点が異なります。

CICS 管理サービスをホスティングする **CICS JVMSERVER** の **z/OS Connect Enterprise Edition (**互換モード**)**
: このオプションは、z/OS Connect のいずれのバージョンでも使用可能です。以前に CICS JSON サービス・テクノロジーでデプロイされたサービスが z/OS Connect に集約されるハイブリッド・モードが提供されます。WSBind ファイルが、DFHLS2JS プロシージャーまたは DFHJS2LS JCL プロシージャーを使用して作成され、WEBSERVICE リソースとして CICS にデプロイされますが、CICS によって z/OS Connect にインストールされます。この構成オプションによって、より古いスタイルの JSON サービスに z/OS Connect Enterprise Edition の多くの利点がもたらされますが、機能に一部の制限や違いがあります。

z/OS Connect 機能の比較

このセクションでは、z/OS Connect Enterprise Edition と z/OS Connect for CICS 1.0 の機能を比較します。z/OS Connect for CICS 1.0 に先行するため、Java Pipelines for JSON (または Feature Pack for Mobile Extensions) に関する比較も含まれています。

表 1. z/OS Connect Enterprise Edition、*z/OS Connect for CICS 1.0*、および *Feature Pack for Mobile Extensions* の機能の比較

機能	**z/OS Connect Enterprise Edition**	**z/OS Connect for CICS 1.0**	**Java Pipelines for JSON (**または **Feature Pack for Mobile Extensions)**
標準の z/OS Connect 機能に対するサポート	あり。サービス・ディスカバリーでは、追加の構成が必要です。	あり。サービス・ディスカバリーでは、追加の構成が必要です。	No

表 1. *z/OS Connect Enterprise Edition*、*z/OS Connect for CICS 1.0*、および *Feature Pack for Mobile Extensions* の機能の比較 (続き)

機能	**z/OS Connect Enterprise Edition**	**z/OS Connect for CICS 1.0**	**Java Pipelines for JSON (**または **Feature Pack for Mobile Extensions)**
CICS プログラムを JSON Web サービスとして有効化	はい	はい	はい
RESTful JSON サービスのサポート	はい	はい	はい
RESTful API サービスのサポート	はい	いいえ	No
リモート JSON サービスの開始のサポート	V3: あり。V2: なし、DFHJSON を使用してください。	なし。DFHJSON を使用してください。	いいえ (DFHJSON を使用)
DFHLS2JS および DFHJS2LS アシスタントを使用する CICS データ変換の関するサポート	あり。注 1 を参照。	はい	はい
SAR ベース・サービスのデプロイメントのサポート	V3: オプション。V2: いいえ。	No	No
JSON スキーマ仕様のサポート	はい (draft-04)	はい (draft-04)	はい (draft-04)
JSON 変換用のパーサー技術の選択	はい	はい	No
PIPELINE ハンドラー・プログラムのサポート	No	No	はい
インターセプター・プログラムのサポート	あり。z/OS Connect インターセプターを使用	あり。z/OS Connect インターセプターを使用	あり。Axis2 ハンドラーを使用
CICS 制御コンテナーへのアプリケーション・アクセス	はい	はい	はい
CICS PIPELINE リソースを使用した WSBind デプロイメント	V3: オプション。V2: はい。注 2 を参照。	はい	はい
URIMAP リソースおよび WEBSERVICE リソースの構成	V3: オプション。V2: はい。注 3 を参照。	はい	はい
ネットワーク構成	WebSphere Liberty を使用	WebSphere Liberty を使用	TCPIPSERVICE リソースを使用

表 1. *z/OS Connect Enterprise Edition*、*z/OS Connect for CICS 1.0*、および *Feature Pack for Mobile Extensions* の機能の比較 (続き)

機能	**z/OS Connect Enterprise Edition**	**z/OS Connect for CICS 1.0**	**Java Pipelines for JSON (**または **Feature Pack for Mobile Extensions)**
セキュリティー構成	WebSphere Liberty および CICS セキュリティーを使用	WebSphere Liberty および CICS セキュリティーを使用	CICS の使用
Transport Layer Security (TLS) のサポート	はい	はい	はい
統計、モニター、ユーザー出口、その他の診断インフラストラクチャー	CICS の WebSphere Liberty と整合性あり	CICS の WebSphere Liberty と同じ	CICS のパイプラインと同じ
JVM 環境	Liberty JVM サーバー (z/OS Connect 専用に分離することが望ましい)	Liberty JVM サーバー (z/OS Connect 専用に分離することが望ましい)	**JAVA_PIPELINE=YES** で構成された非 OSGi JVM (Java Pipelines for JSON 専用に分離することが望ましい)

注**:**

1. CICS TS で提供される CICS サービス・プロバイダーを使用する場合、この機能はサービスでサポートされます。API の場合、z/OS Connect Enterprise Edition で提供されるアシスタントを使用してください。

   z/OS Connect Enterprise Edition V3.0 で提供される CICS サービス・プロバイダーを使用する場合、この機能はサポートされません。z/OS Connect Enterprise Edition API Toolkit を使用するか、ビルド・ツールキットを使用してサービスを作成してから API ツールキットを使用して API を作成します。
2. CICS TS で提供される CICS サービス・プロバイダーを使用する場合、この機能はサービスでサポートされますが、API のデプロイには追加の成果物が関与します。詳しくは、z/OS Connect Enterprise Edition V3.0 の製品資料を参照してください。

   z/OS Connect Enterprise Edition V3.0 で提供される CICS サービス・プロバイダーを使用する場合、この機能はサービスでサポートされません。z/OS Connect Enterprise Edition サービス・ディレクトリーを使用します。API のデプロイには追加の成果物が含まれます。詳しくは、z/OS Connect Enterprise Edition V3.0 の製品資料を参照してください。
3. この機能は、CICS TS で提供される CICS サービス・プロバイダーを使用する場合にのみ、サービスでサポートされます。z/OS Connect Enterprise Edition V3.0 で提供される CICS サービス・プロバイダーでは、URIMAP の手動作成がサポートされます。例えば、URIMAP を使用して、デフォルトの CICS トランザクション (CJSA) をオーバーライドできます。

z/OS Connect for CICS の機能

CICS に統合された z/OS Connect の機能は、他の環境における z/OS Connect の機能と異なる部分があります。z/OS Connect for CICS は、CICS へのローカル・アクセス向けに最適化されており、標準のCICS サービスを使用します。

z/OS Connect for CICS には、WebSphere Liberty for z/OS での z/OS Connect と同じ機能がありますが、いくつかの相違点もあります。

- WSBind ファイルは、CICS の提供する DFHLS2JS および DFHJS2LS ツールを使用して生成できます。
- WSBind ファイルは、**WEBSERVICE** および **URIMAP** リソースを使用してz/OS Connect for CICS のインフラストラクチャーにデプロイされます。これにより、z/OS Connect for CICS の運用管理は、他の CICS Web サービスと同じようなものになります。
- CICS でホストする JSON は、最適化されたローカル接続を介して CICS プログラムと対話できます。
- JSON サービスは、ボトムアップ開発とトップダウン開発のどちらの方法でも開発できます。
- RESTful スタイルの JSON サービスを、CICS に格納できます。
- 統合された z/OS Connect for CICS を使用して、IMS™ やバッチなどの他の z/OS サブシステムのアセットと直接対話することはできません。
- サービスのインストールに PIPELINE スキャン・コマンドを使用した場合は、WebSphere Liberty Profile の `server.xml` ファイルで CICS に対するサービスごとの構成を変更する必要はありません。ただし、`server.xml` でのサービスごとの構成はサポートされています。
- z/OS Connect および z/OS Connect for CICS のいずれも、URI のルート・コンテキストを取得します。z/OS Connect for CICS は、専用の Liberty JVM サーバー環境にデプロイすることをお勧めします。

関連情報:

z/OS Connect for CICS の構成

z/OS Connect for CICS のセキュリティー

第 2 章 CICS での Web サービスの構成

Web サービスをサポートするように CICS を構成することができます。この場合、CICS アプリケーションは Web サービス・リクエスターにも Web サービス・プロバイダーにもなることができます。 CICS は、バイナリー添付ファイルや Web サービス・アドレッシングをはじめ、さまざまな Web サービス規格をサポートしています。 WebSphere MQ または HTTP からの Web サービス要求を受け入れ、WSRR から WSDL ファイルを取得するように CICS を構成することもできます。

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。

手順

1. PL/I の言語環境プログラム・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。 詳しくは、言語環境プログラム・サポートのインストールを参照してください。
2. z/OS Support for Unicode を活動化します。 z/OS 変換サービスを使用可能にして、CICS を使用して SOAP メッセージとアプリケーション・プログラム間で実行するデータ変換を指定する変換イメージをインストールする必要があります。 詳しくは、z/OS Unicode Services ユーザーズ・ガイドおよび解説書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 PIPELINE、WEBSERVICE、URIMAP、および TCPIPSERVICE リソース。

PIPELINE

PIPELINE リソース定義は、あらゆる Web サービスに必要です。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 リソース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 リソース定義はハンドラーとその構成の XML 記述を格納する z/OS UNIX 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 定義は、CONFIGFILE 属性に指定されているパイプライン構成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

© Copyright IBM Corp. 2005, 2018

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 リソース定義のピックアップ・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 リソースは、リソース定義内の 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

要求を処理する他のリソース (PIPELINE リソースなど) にインバウンド Web サービス要求の URI をマップする情報が URIMAP 定義に含まれる場合、この定義はサービス・プロバイダーで必要になります。 URIMAP リソース定義はサービス・リクエスターのユーザー ID 情報が HTTP 認証ヘッダーに組み込まれてサービス・プロバイダーに渡されるように指定しているため、この URIMAP 定義は、HTTP 基本認証を使用する場合にも必要となります。

WSDL ディスカバリーのために、サービス・プロバイダーにオプションの 2 番目の URIMAP 定義が存在する場合があります。 この URIMAP リソース定義には、Web サービスと関連付けられた WSDL 文書のインバウンド要求の URI をマップする情報が含まれています。

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、CICS は通常のリソース定義メカニズムを備えていますが、通常、ピックアップ・ディレクトリーのスキャン時に URIMAP リソースが自動的に作成されます。 このスキャンは、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。 WEBSERVICE リソースを特定の URI に関連付けるための情報を CICS に提供する URIMAP リソースは、必須リソースです。このリソースの属性は、ピックアップ・ディレクトリー内の Web サービス・バインディング・ファイルによって指定されます。 WSDL アーカイブ・ファイルまたは WSDL 文書を特定の URI に関連付けるための情報を CICS に提供する URIMAP リソースは、オプション・リソースであり、ピックアップ・ディレクトリーに WSDL ファイルまたは WSDL アーカイブ・ファイルが存在する場合に作成されます。Web サービス・プロバイダ

ーの URIMAP リソース作成について詳しくは、Web サービス・アシスタントを使用した Web サービス・プロバイダーの作成 を参照してください。

サービス・リクエスターに関しては、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、CICS が URIMAP リソースを自動的に作成することはありません。サービス・リクエスターは、要求時に URIMAP リソースを使用するようには求められていません。アプリケーション・プログラムでアウトバウンド要求の URI を直接指定できます。 ただし、クライアント要求のために URIMAP リソースを作成し、サービス・リクエスターが URIMAP リソースを使用して URI を提供する場合には、以下の利点があります。

- システム管理者が接続のエンドポイントへの変更を管理できるため、サービス・プロバイダーの URI が変更されても、アプリケーションを再コンパイルする必要がありません。
- CICS により、URIMAP リソースで開かれた接続を使用後に開いたままにし、その接続をアプリケーションの後続の要求、または同じサービスを呼び出す別のアプリケーションが再利用できるようにプールに入れておくことを選択できます。 接続プールを使用できるのは、SOCKETCLOSE 属性が設定されている URIMAP リソースを指定した場合だけです。接続プールのパフォーマンス上の利点について詳しくは、HTTP クライアントのパフォーマンスのための接続プールを参照してください。

特定の方法で URIMAP リソース属性を構成することで、インバウンド要求を直接接続されたユーザー・トランザクションによって処理し、Web 接続タスクをバイパスできるようになる場合があります。詳しくは、直接接続されたユーザー・トランザクションにより HTTP 要求が処理されるを参照してください。

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、注 1 を参照)	はい (ただし、注 1 を参照)	注 2 を参照

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	No	はい	No	はい	注 2 を参照
リクエスター	はい	はい	はい	注 3 を参照	No
リクエスター	No	はい	No	3	No

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 2 つの URIMAP リソースを自動的に作成することができます。最初の URIMAP リソースは必須であり、WEBSERVICE リソースを特定の URI に関連付けるための情報を CICS に提供します。2 番目の URIMAP リソースはオプションであり、WSDL アーカイブ・ファイルまたは WSDL 文書を特定の URI に関連付けるための情報を CICS に提供します。これにより、外部リクエスターはその URI を使用して、WSDL アーカイブ・ファイルまたは WSDL 文書を見つけることができます。 PIPELINE のピックアップ・ディレクトリーのスキャンは、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、TCPIPSERVICE リソースは必要ありません。
3. サービス・リクエスターでは URIMAP リソースはオプションであり、CICS Web サービス・アシスタントによって自動的に生成されることはありません。サービス・リクエスターで使用するために独自の URIMAP リソースを定義する場合、接続プールを実装し、サービス・プロバイダーの URI への変更を管理できます。

特定の方法で TCPIP リソース属性を構成することで、インバウンド要求を直接接続されたユーザー・トランザクションによって処理し、Web 接続タスクをバイパスできるようになる場合があります。詳しくは、直接接続されたユーザー・トランザクションにより HTTP 要求が処理されるを参照してください。

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。 各 Web サービス・ファイルまたはリソースは、他のタイプの 1 つ以上の CICS リソースと関連付けられています。

表 2. 各 *Web* サービス・ファイルおよびリソースと関連付けられている他の *CICS* リソース

Web サービス・ファイルまたはリソース	関連付けられているリソース
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	• PIPELINE リソースを参照する複数の URIMAP リソース。 • PIPELINE リソースを参照する複数の WEBSERVICE リソース。 • PIPELINE リソースのピックアップ・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。
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。 サービス・プロバイダーにさらに URIMAP リソースを定義できます。また、サービス・リクエスターに URIMAP リソースを定義できます。 • 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。 必要に応じて、さらに WEBSERVICE リソースを定義できます。
WEBSERVICE	• 複数の URIMAP リソース。 サービス・プロバイダーのバインディング・ファイルから WEBSERVICE リソースが自動的に生成される場合、CICS によって、対応する URIMAP リソースが 1 つ生成されます。 サービス・プロバイダーにさらに URIMAP リソースを定義できます。また、サービス・リクエスターに URIMAP リソースを定義できます。
URIMAP	• URIMAP リソースで明示的に指定される場合は、1 つの TCPIPSERVICE リソースのみ
TCPIPSERVICE	• 多数の URIMAP リソース

Web サービス・ディスカバリー

プロバイダー・モード Web サービスに関連付けられた WSDL 文書は、Web に自動的に公開されます。

規則は、Web サービスの WSDL がリモート・クライアント (通常は、Web ブラウザーを使用したアプリケーション開発者) によって、接尾部に `?wsdl` が付いた Web サービスの URI を使用して照会されるのを許可する、 Web サービスのホスティング環境にあります。この規則により、正式な WSDL リポジトリーがなくても関係者に WSDL を配布するのが容易になります。この規則は、CICS に実装されています。

例えば、Web サービスが CICS でホストされ、以下の URI の下で公開される場合です。

```
http://www.example.org:1234/example/WebService
```

関連付けられた WSDL 文書は、Web ブラウザーを使用して以下の URI を要求することにより、リカバリーできました。

```
http://www.example.org:1234/example/WebService?wsdl
```

URIMAP リソースを使用したディスカバリーのために、サービス・プロバイダーの WSDL 文書をパブリッシュできます。各 PIPELINE リソースをインストールする際、CICS は、PIPELINE リソースの 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に、WSDL アーカイブ・ファイルまたは WSDL 文書のいずれかが含まれている場合、2 番目の URIMAP リソースがインストールされます。この新しい URIMAP リソースは、WSDL アーカイブ・ファイルまたは WSDL 文書と特定の URI を関連付ける情報を CICS に提供するので、外部リクエスターは URI を使用して WSDL アーカイブ・ファイルまたは WSDL 文書をディスカバーできます。この URI は、WEBSERVICE と関連した URI と同じパスで、サフィックス `?wsdl` が追加されたものになります。

WSDL アーカイブ・ファイルは、1 つ以上の WSDL 文書を含みます。ピックアップ・ディレクトリーに WSDL アーカイブ・ファイルおよび WSDL 文書が含まれる場合、URI は WSDL アーカイブだけを戻します。サポートされているアーカイブ・ファイル・フォーマットは、.zip ファイル・タイプです。SPI および CEMT を使用して、WSDL アーカイブ・ファイルまたは WSDL 文書をディスカバーすることもできます。WSDL アーカイブ・ファイルの WSDL 文書を SOAP メッセージの検証に使用できます。

WebSphere MQ トランスポートを使用するための CICS の構成

CICS で WebSphere MQ トランスポートを SOAP 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。

このタスクについて

注: JSON Web サービス用に Websphere MQ トランスポートを使用することはできません。

手順

1. IBM MQ ライブラリー *thlqual*.SCSQAUTH を CICS プロシージャーの STEPLIB 連結に組み込みます。正しいコードが確実に使われるようにするために、CICS ライブラリーの後にこのライブラリーを組み込んでください。*thlqual* は、IBM MQ ライブラリーの高位修飾子です。
2. 以下の IBM MQ ライブラリーを CICS プロシージャーの DFHRPL 連結に組み込みます。正しいコードが確実に使われるようにするために、CICS ライブラリーの後にこのライブラリーを組み込んでください。

   thlqual.SCSQCICS

   thlqual.SCSQLOAD

   thlqual.SCSQAUTH

thlqual は、IBM MQ ライブラリーの高位修飾子です。 CICS-MQ API 交差出口 (CSQCAPX) を使用している場合は、プログラムのロード・モジュールを含んでいるライブラリーの名前も追加してください。 SCSQCICS ライブラリーは、IBM MQ に付属のサンプルを実行する場合に限って必要です。 そうでない場合は、CICS プロシージャーからこれを除去することができます。

3. CICS 領域用の MQCONN リソースを定義してインストールします。 MQCONN リソースは、CICS と IBM MQ との接続の属性を指定します。これには、接続のデフォルト IBM MQ キュー・マネージャーまたはキュー共用グループの名前が含まれます。 説明は、MQCONN リソースの定義とインストールを参照してください。
4. CICS の初期化で CICS と WebSphere MQ との接続を自動的に開始するには、CICS システム初期設定パラメーター **MQCONN=YES** を指定します。詳細については、MQCONN システム初期設定パラメーターを参照してください。
5. リモート CICS システムとの領域間通信 (IRC) を行う CICS システムで CICS-MQ アダプターを使用する場合には、CICS システム初期設定パラメーター IRCSTRT=YES を指定することにより、アダプター開始前に IRC 機能を必ず OPEN してください。 IRC アクセス方式が仮想記憶間として定義されている場合 (つまり ACCESSMETHOD(XM) である場合)、IRC 機能が OPEN している必要があります。
6. ご使用のキュー・マネージャーおよび CICS で使用されるコード化文字セット ID (CCSID)、および UTF-8 と UTF-16 のコード・ページが、z/OS 変換サービスに対して構成されていることを確認します。 CICS コード・ページは、**LOCALCCSID** システム初期化パラメーターで指定されます。
7. 次のようにして CICS CSD を更新します。
   a. 以前のリリースの CICS との間で CSD を共用しない場合は、グループ CSQCAT1 と CSQCKB、およびこれらのグループ (またはグループ項目) のコピーを CSD からすべて削除します。また、CKQQ TDQUEUE をグループ CSQCAT1 から削除する必要もあります。 これで、CKQQ の定義が CICS CSD グループ DFHDCTG で提供されるようになります。
   b. 以前の CICS リリースとの間で CSD を共用する場合は、CSQCAT1 と CSQCKB、およびこれらのグループ (またはその内容) のコピーが CICS TS 4.1 または CICS TS 3.2 用にインストールされていないことを確認してください。また、CKQQ TDQUEUE をグループ CSQCAT1 から削除する必要もあります。 これで、CKQQ の定義が CICS CSD グループ DFHDCTG で提供されるようになります。 CICS TS 3.2 より前の CICS TS リリースの場合、グループ DFHMQ を指定変更して必要な定義を正しくインストールするために、DFHLIST のインストール後に CSQCAT1 および CSQCKB グループをグループ・リストに含めてインストールしてください。
8. デッド・レター・キュー、デフォルト伝送キュー、および CICS-MQ アダプター・オブジェクトに関する IBM MQ 定義を更新します。 サンプル CSQ4INYG を使用できますが、CICS 領域の MQINI リソース定義内のデフォルト開始キュー名に一致するよう、開始キュー名を変更する必要があるかもしれません。 このメンバーをキュー・マネージャー始動プロシージャーの CSQINP2 DD 連結で使用できます。または、必要な DEFINE コマンドを発行するために、CSQUTIL ユーティリティーの COMMAND 関数の入力としてこ

のメンバーを使用することもできます。 CSQUTIL ユーティリティーを使用すると、それ以降 IBM MQ を再始動するたびにこれらのオブジェクトを再定義する必要がなくなるため、この方法をお勧めします。

IBM MQ トランスポート

CICS は、IBM MQ トランスポートを使用して、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役割で、IBM MQ との間で SOAP メッセージを送受信できます。

サービス・プロバイダーとして、CICS は IBM MQ トリガーを使用して、SOAP メッセージをアプリケーション・キューから処理します。トリガーは、開始キューとローカル・キューを使用することで動作します。 ローカル (アプリケーション) キュー定義には、以下の情報が含まれます。

- トリガー・メッセージが生成される時期についての基準。 例えば、最初のメッセージがローカル・キューに到着したとき、またはローカル・キューに到着するメッセージごとに、など。 CICS SOAP 処理については、最初のメッセージがローカル・キューに到着したときにトリガーが起こるように指定します。

  ローカル・キュー定義では、トリガー・データをターゲット・アプリケーションに渡すように指定することもできます。CICS SOAP 処理 (トランザクション CPIL) の場合は、インバウンド・メッセージで渡されない場合に使用されるデフォルトのターゲット URL を指定します。
- プロセス定義 を識別するプロセス名。 プロセス定義では、メッセージを処理する方法が記述されます。 CICS SOAP 処理の場合は、CPIL トランザクションを指定します。
- トリガー・メッセージが送信される開始キューの名前。

メッセージがローカル・キューに到着すると、キュー・マネージャーがトリガー・メッセージを生成し、指定された開始キューに送信します。 トリガー・メッセージには、プロセス定義からの情報が含まれます。 トリガー・モニターは開始キューからトリガー・メッセージを取り出し、ローカル・キューでメッセージの処理を開始するように CPIL トランザクションをスケジュールします。 トリガーについて詳しくは、タスク・イニシエーターまたはトリガー・モニター (CKTI)を参照してください。

CICS を構成することで、メッセージがローカル・キューに到着すると、トリガー・モニター (IBM MQ 提供) が CPIL トランザクションをスケジュールして、ローカル・キューのメッセージを処理し、 CICS SOAP パイプラインにキューの SOAP メッセージを処理させることができます。

CICS が IBM MQ から受け取る SOAP メッセージへの応答を構成するとき、レポート・オプション MQRO_PASS_CORREL_ID が設定されていない限り、入力メッセージのメッセージ ID で相関 ID フィールドが設定されます。 このレポート・オプションが設定されていれば、相関 ID は入力メッセージから応答に伝搬されます。

サービス・リクエスターとして、アウトバウンド要求で、ターゲット Web サービスへの応答が特定の応答キューで戻るように指定できます。

どちらの場合も、CICS および IBM MQ では、必要なリソースおよびキューを定義するための構成が必要です。

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、要求メッセージを処理するまで要求メッセージを保管する 1 つ以上のローカル・キューと、要求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。

手順

1. 開始キューを定義します。 以下のコマンドを使用します。

   ```
   DEFINE
   QLOCAL('initiation_queue')
   DESCR('description')
   ```

   ここで、*initiation_queue* は、CICS 領域のインストール済み MQMONITOR リソース定義の QNAME 属性に指定されている値と同じか、インストール済み MQCONN リソース定義の INITQNAME 属性に指定されている値と同じです。
2. ローカル要求キューごとに、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('initiation_queue')
   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。
   - *initiation_queue* は、使用される開始キューの名前です。例えば、CICS 領域のインストール済み MQMONITOR リソース定義の QNAME 属性で指定されている開始キューです。
   - *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・サービスの形式は「/string」で、これは URIMAP 定義のパスと突き合わせるために使用されます (例えば /SOAP/test/test1)。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。

3. 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、要求キューの定義時に使用される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 WebSphere 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。

このタスクについて

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。

手順

オプション: 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、

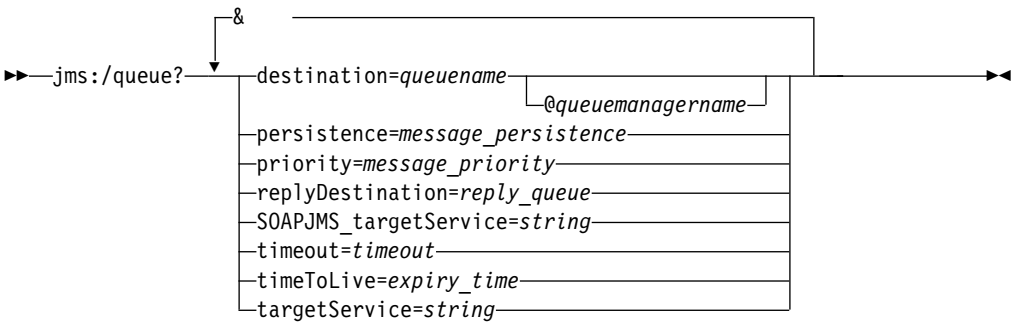
reply_queue は、ローカル・キュー名です。

nnn は、行われる再試行の回数です。

IBM MQ トランスポートの URI

サービス・リクエスターとサービス・プロバイダーの間の通信に IBM MQ を使用するとき、宛先の URI は、宛先をキューとして識別する形式であり、IBM MQ が要求と応答を処理する方法を指定するための情報を含んでいます。

構文



オプション

CICS は以下のオプションを使用します。他の Web サービス・プロバイダーは、ここで説明しない追加のオプションを使用することがあります。 URI 全体がサービス・プロバイダーに渡されます。ただし、CICS は、CICS がサポートしないオプションおよび URI でコーディングされているオプションを無視します。 CICS はオプション名の大/小文字を区別しません。 ただし、このスタイルの URI をサポートするその他の実装環境には、大/小文字を区別するものがあります。

destination=*queuename* [@*queuemanagername*]

queuename は宛先キュー・マネージャーの入力キューの名前です。

queuemanagername は宛先キュー・マネージャーの名前です。

persistence=*message_persistence*

以下のいずれかのオプションを指定します。

0 永続性はデフォルトのキューの永続性によって定義されます。

1 メッセージは永続ではありません。

2 メッセージは永続です。

このオプションが指定されないか、誤って指定されると、デフォルトのキューの永続性が使用されます。

priority=*message_priority*

メッセージ優先順位を指定します。 CICS は、メッセージ優先順位として 0 から 9 の範囲の整数値をサポートします。ここで、9 は最も優先順位が高いメッセージに割り当てられ、0 は最も優先順位が低いメッセージに割り当てられます。別の方法として、**-1** を指定すると、宛先キューに定義されているデフォルトの優先順位が使用されます。

replyDestination=*reply_queue*

応答メッセージに使用されるキューを指定します。 このオプションが指定されていない場合、CICS は応答メッセージに動的キューを使用します。このオプションを使用する前に QLOCAL オブジェクトに応答キューを定義する必要があります。

SOAPJMS_targetService=*string*

ターゲット・サービスを識別します。 CICS がサービス・プロバイダーである場合は、ターゲット・サービスの形式は '/string' になります。なぜなら、CICS はこれをパスとして使用して URIMAP との突き合わせを試みるためで

す。このオプションを指定しない場合は、サービス・プロバイダーにおける入力キューの TRIGDATA に指定された値が使用されます。

timeout=*timeout*
サービス・リクエスターが応答を待機するときにタイムアウトになるまでの長さ (ミリ秒単位)。 値ゼロが指定されるか、このオプションが除外される場合は、要求はタイムアウトになりません。

timeToLive=*expiry-time*
要求の有効期限時刻をミリ秒単位で指定します。 このオプションが指定されないか、誤って指定されると、要求の有効期限が切れません。

targetService=*string*
ターゲット・サービスを識別します。 CICS がサービス・プロバイダーである場合は、ターゲット・サービスの形式は `'/string'` になります。なぜなら、CICS はこれをパスとして使用して URIMAP との突き合わせを試みるためです。このオプションを指定しない場合は、サービス・プロバイダーにおける入力キューの TRIGDATA に指定された値が使用されます。

例

この例は IBM MQ トランスポートの URI を示しています。

```
jms:/queue?destination=queue01@cics007&timeToLive=10&replyDestination=rqueue05&targetService=/myservice
```

`"connectionFactory"` および `"initialContextFactory"` については、IBM MQ 製品資料を参照してください。

永続メッセージをサポートするための CICS の構成

CICS は、WebSphere MQ トランスポート・プロトコルを使用した永続メッセージの、CICS 領域に配置された Web サービス・プロバイダー・アプリケーションへの送信をサポートしています。

このタスクについて

CICS は、ビジネス・トランザクション・サービス (BTS) を使用して、CICS システム障害の際に永続メッセージが確実に回復されるようにします。 これを正しく機能させるためには、以下のステップに従います。

手順

1. IDCAMS (データ操作ユーティリティー) を使用して、MVS™ へのローカルの要求キューとリポジトリー・ファイルを定義します。 ファイル定義のために、STRINGS に適切な値を指定する必要があります。 デフォルト値 1 では十分ではないと思われるため、10 を使用することをお勧めします。
2. CICS に対するローカルの要求キューとリポジトリー・ファイルを定義します。CICS に対するローカルの要求キューを定義する方法の詳細は、 55 ページの『サービス・プロバイダーでのローカル・キューの定義』で説明しています。ファイル定義に、STRINGS に適切な値を指定する必要があります。 デフォルト値 1 では十分ではないと思われるため、10 を使用することをお勧めします。
3. リポジトリー・ファイル名を FILE オプションの値として使用して、DFHMQSOA という名前の PROCESSTYPE リソースを定義します。

4. 永続メッセージの処理中に、最初の暗黙的な同期点が要求される前にプログラムが **EXEC CICS SYNCPOINT** コマンドを必ず発行することを確認してください。例えば **EXEC CICS CREATE TDQUEUE** のような SPI コマンドを使用すると、暗黙的に同期点が取られます。 **EXEC CICS SYNCPOINT** コマンドを発行すると、永続メッセージが正常に処理されたことを確認できます。プログラムが暗黙的に同期点を取ることを試みる前に明示的に同期点を要求しない場合、ASP7 異常終了が発行されます。

タスクの結果

次のタスク

片方向の要求メッセージの場合は、Web サービスが異常終了またはバックアウトすると、トランザクションまたはプログラムが障害が発生している要求を再試行したり障害を適切に報告したりするための十分な情報が保持されます。 このようなリカバリー・トランザクションまたはプログラムを提供する必要があります。詳細については、『永続メッセージの処理』を参照してください。

永続メッセージの処理

Web サービス要求が WebSphere MQ 永続メッセージで受信されると、CICS は、プロセス・タイプが DFHMQSOA である固有の BTS プロセスを作成します。 インバウンド要求に関連するデータは、プロセスに関連付けられた BTS データ・コンテナー内に取り込まれます。

プロセスは、非同期で実行されるようにスケジュールに入れられます。 Web サービスが正常に完了してコミットすると、CICS は BTS プロセスを削除します。 これには、SOAP 障害が Web サービス・リクエスターに生成され戻される場合が含まれます。

エラー処理

必要な BTS プロセスの作成時にエラーが発生すると、Web サービス・トランザクションは異常終了し、インバウンド Web サービス要求は処理されません。 BTS が使用不可の場合は、メッセージ DFHPI0117 が発行され、CICS は、既存のチャネル・ベースのコンテナー・メカニズムを使用して、BTS なしで続行します。

Web サービスが開始または処理を完了する前に CICS 障害が発生すると、BTS リカバリーにより、CICS の再起動時にプロセスのスケジュールが変更されます。

Web サービスが異常終了してバックアウトすると、BTS プロセスには、ABENDED 状況で完了したというマークが付きます。 応答を必要とする要求メッセージの場合は、SOAP 障害が Web サービス・リクエスターに戻されます。 BTS プロセスは取り消され、CICS は、失敗した要求に関する情報を保持しません。 CICS は、一時データ・キュー CSBA ではメッセージ DFHBA0104 を発行し、一時データ・キュー CPIO ではメッセージ DFHPI0117 を発行します。

片方向メッセージの場合、障害に関する情報をリクエスターに戻す方法はないため、BTS プロセスは COMPLETE ABENDED 状態を保ちます。 CICS は、一時データ・キュー CSBA ではメッセージ DFHBA0104 を発行し、一時データ・キュー CPIO では DFHPI0116 を発行します。

CBAM トランザクションを使用して COMPLETE ABENDED プロセスを表示することができます。または、リカバリー・トランザクションを指定して、DFHMQSOA の COMPLETE ABENDED プロセスをチェックし、適切な処置をとることができます。

例えば、リカバリー・トランザクションで以下のことが可能です。

1. **RESET ACQPROCESS** コマンドを使用して BTS プロセスをリセットする。
2. **RUN ASYNC** コマンドを発行して、障害がある Web サービスを再試行する。 プロセスでの別のデータ・コンテナーに再試行カウントを保持して、障害が繰り返されるのを回避することができます。
3. 関連する以下のデータ・コンテナー内の情報を使用して、問題を報告する。

   DFHMQORIGINALMSG データ・コンテナーには、WebSphere MQ から受信したメッセージが含まれ、これには RFH2 ヘッダーが含まれている場合があります。

   DFHMQMSG データ・コンテナーには、RFH2 ヘッダーが除去された WebSphere MQ メッセージが含まれます。

   DFHMQDLQ データ・コンテナーには、元のメッセージに関連付けられた送達不能キューの名前が含まれます。

   DFHMQCONT データ・コンテナーには、元のメッセージの **MQ GET** に関連する WebSphere MQ MQMD 制御ブロックが含まれます。

Web サービス・アシスタントと WSRR の間の相互運用性

CICS Web サービス・アシスタントは、IBM WebSphere Service Registry and Repository (WSRR) と相互運用することができます。 WSRR を使用して、要求している Web サービスをより短時間で見つけ、提供している Web サービスのバージョン管理を実施します。

DFHLS2WS および DFHWS2LS には、WSRR と相互運用するためのパラメーターが含まれています。 また、DFHLS2WS には、WSRR の WSDL 文書に独自のカスタマイズ・メタデータを追加するためのオプション・パラメーターも含まれています。

Web サービス・アシスタントと WSRR の間の通信での機密保護を実現するには、SSL (Secure Socket Level) 暗号化を使用することができます。 DFHLS2WS および DFHWS2LS には、SSL 暗号化を使用するためのパラメーターが含まれています。

Web サービス・アシスタントと WSRR で SSL を使用するには、『Web サービス・アシスタントと WSRR で SSL を使用する方法の例』を参照してください。

Web サービス・アシスタントと WSRR で SSL を使用する方法の例

Secure Sockets Layer (SSL) 暗号化を使用することにより、Web サービス・アシスタントと IBM WebSphere Service Registry and Repository (WSRR) サーバーの間で安全に相互運用することができます。 SSL 暗号化を使用するには鍵ストアとトラストストアが必要です。さらに、Web サービス・アシスタントでいくつかのパラメーターを指定する必要もあります。

このタスクについて

Web サービス・アシスタントと WSRR の対話のために SSL 暗号化を使用するには、以下の手順を完了します。

手順

1. 秘密鍵および公開鍵証明書 (PKC) のための鍵ストアを作成します。
   a. IBM 鍵管理ユーティリティー (iKeyman) などの鍵構成プログラムを使って鍵ストアを作成できます。
   b. 作成した鍵ストアの完全修飾名を、DFHWS2LS または DFHLS2WS の **SSL-KEYSTORE** パラメーターで指定します。
   c. オプション: 作成した鍵ストアのパスワードを、DFHWS2LS または DFHLS2WS の **SSL-KEYPWD** パラメーターで指定します。
2. すべてのトラステッド・ルート認証局 (CA) 証明書のためのトラストストアを作成します。 これらの証明書は、インバウンド公開鍵証明書の信用を確立するために使用されます。
   a. IBM 鍵管理ユーティリティー (iKeyman) などの鍵構成プログラムを使ってトラストストアを作成できます。
   b. 作成したトラストストアの完全修飾名を、DFHWS2LS または DFHLS2WS の **SSL-TRUSTSTORE** パラメーターで指定します。
   c. オプション: 作成したトラストストアのパスワードを、DFHWS2LS または DFHLS2WS の **SSL-TRUSTPWD** パラメーターで指定します。
3. Web サービス・アシスタントが SSL 暗号化を使って WSRR と通信できることをテストします。
   a. Web サービス・アシスタントと WSRR の通信をテストするには、IBM WebSphere Application Server に付属のサンプル・ファイルを使用できます。
      - WebSphere Application Server に付属のサンプル鍵ストアは `DummyClientKeyFile.jks` および `DummyServerKeyFile.jks` です。
      - WebSphere Application Server に付属のサンプル・トラストストアは `DummyClientTrustFile.jks` および `DummyServerTrustFile.jks` です。
   b. 鍵ストアおよびトラストストアのサンプル・ファイル内の鍵を置き換えます。 これらは WebSphere Application Server に付属している鍵であり、セキュリティーのために置き換える必要があります。

タスクの結果

これで、Web サービス・アシスタントは SSL 暗号化を使用してネットワークで WSRR と安全に通信できます。

Web サービス・インフラストラクチャーの作成

Web サービスを CICS に配置するには、必要なトランスポート・インフラストラクチャーを作成して、Web サービス要求を処理する 1 つ以上のパイプラインを定義する必要があります。 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。

Web サービス・インフラストラクチャー

CICS 領域内の CICS アプリケーションは、Web サービス・パイプラインを使用して、その領域外のアプリケーションへのサービスの提供、または領域外のアプリケーションからのサービスの要求を行えます。 CICS がサービス・プロバイダーである場合、CICS アプリケーションは外部アプリケーションへサービスを提供します。 CICS がサービス・リクエスターである場合、外部アプリケーションが CICS アプリケーションへサービスを提供します。 zEnterprise® Application Assist Processor (zAAP) を使用できる場合には、それを使用するように Web サービス・パイプラインを構成することもできます。

サービス・プロバイダーとしての CICS

CICS が外部サービス・リクエスターへサービスを提供する場合、CICS がサービス要求を受け取り、それをパイプラインを介してターゲット・アプリケーション・プログラムへ渡す必要があります。 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。

63 ページの図 15 は、CICS が Java パイプラインを使用するサービス・プロバイダーである場合に、外部サービス・リクエスターからの要求を処理するために必要なアーキテクチャーおよびリソースの構成例を示しています。

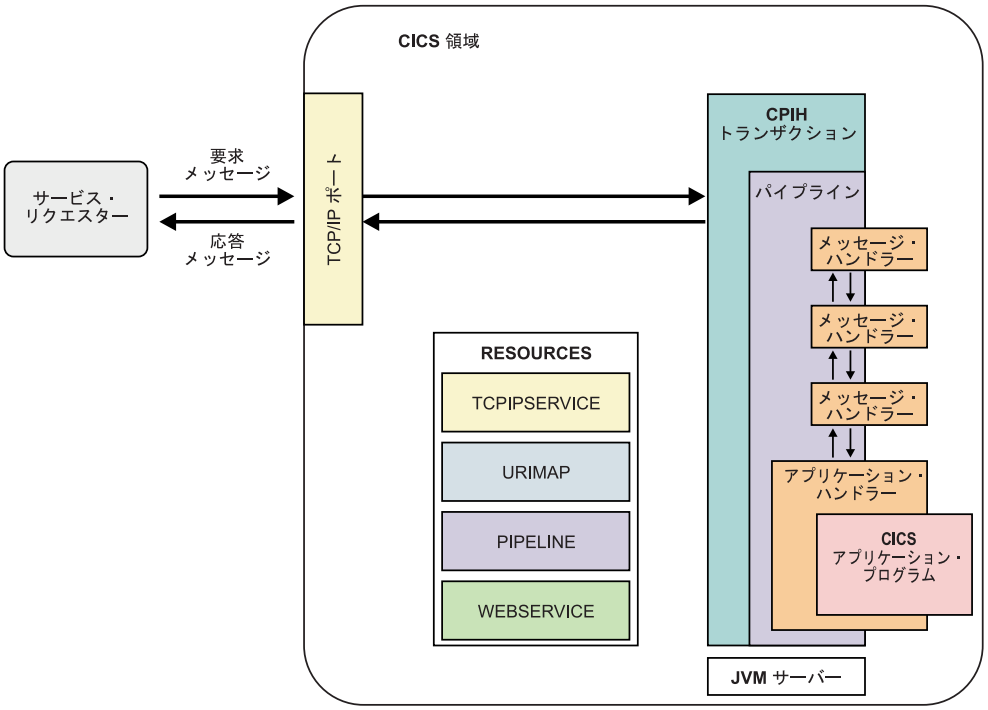


図 *15.* サービス・プロバイダーのアーキテクチャーおよびリソース

要求を処理するために、CICS は以下の操作を実行する必要があります。

1. サービス・リクエスターからの要求を受け取る。

   TCPIPSERVICE リソースは着信要求のポートを指定します。 このポートは、CICS 提供のソケット・リスナー・トランザクション (CSOL) によってモニターされます。

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。

   適切なポートで要求メッセージを受け取ると、URIMAP リソース定義がスキャンされ、その中で USAGE 属性の設定が PIPELINE になっており、かつ PATH 属性の設定が要求で検出された URI になっている URIMAP 定義が検出されます。 適切な URIMAP 定義が見つかった場合、その URIMAP 定義の PIPELINE および WEBSERVICE の各属性で指定された PIPELINE および WEBSERVICE 定義が使用されます。 URIMAP 定義の TRANSACTION 属性によって、パイプラインを処理するために接続されているトランザクションの名前が判別されます。 デフォルトでは、CPIH トランザクションが使用されます。 URIMAP 定義では、使用する PIPELINE および WEBSERVICE リソースも識別されます。 これらのリソースは、その CICS が実行する処理を制御します。

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。

   パイプラインのメッセージ・ハンドラー、およびアプリケーション・ハンドラーは、サービス・プロバイダーのアプリケーション・プログラムが受け入れるアプリケーション言語構造に要求メッセージを変換します。 このプログラムはこの入力を処理し、アプリケーション・ハンドラーに応答を返します。

4. アプリケーション・プログラムによって戻されるデータを使用して応答を作成し、応答をサービス・リクエスターに送信する。

   アプリケーション・ハンドラーおよびメッセージ・ハンドラーは、サービス・プロバイダー・アプリケーションから受け取った応答メッセージを、元の要求の形式のメッセージに変換します。 このメッセージはサービス・リクエスターに戻されます。

パイプラインが適切に構成されていれば、パイプライン内の処理の一部を zEnterprise Application Assist Processor (zAAP) を使用して実行できます。詳しくは、 65 ページの『Java ベースの SOAP パイプライン』を参照してください。

サービス・リクエスターとしての CICS

CICS が外部サービスを呼び出す場合、アプリケーション・プログラムは、パイプラインを介して渡される要求をターゲット・サービスに送信します。 サービスからの応答は、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。

図 16 は、 CICS 領域外のサービス・プロバイダーからのデータに関して、 CICS アプリケーション・プログラムからの要求を Java パイプラインを使用して処理するために必要なアーキテクチャーおよびリソースの構成例を示しています。

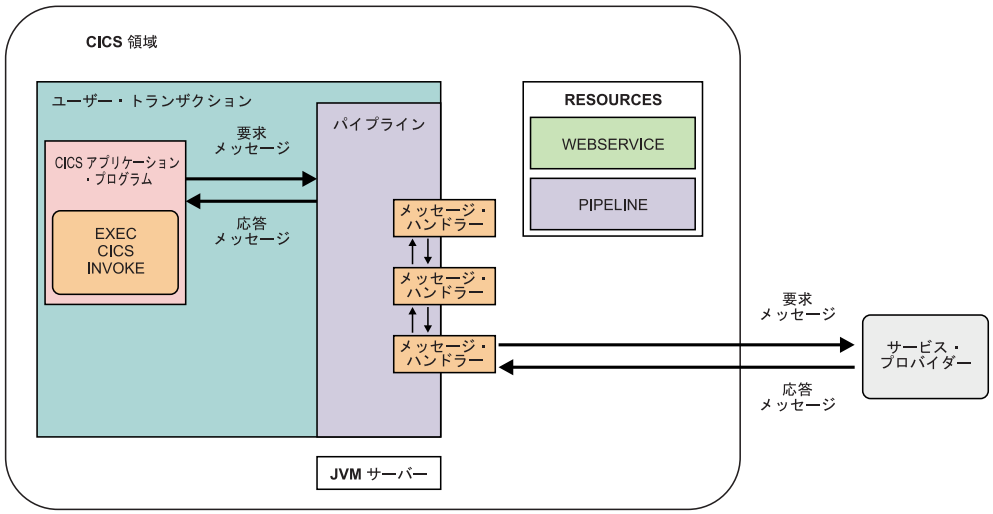


図 *16.* サービス・リクエスターのアーキテクチャーおよびリソース

要求を処理するために、CICS は以下の操作を実行する必要があります。

1. アプリケーション・プログラムによって提供されるデータを使用して要求を作成する。

   CICS アプリケーション・プログラムが CICS 領域外のサービス・プロバイダーへの要求を開始する場合、リクエスター・アプリケーションは EXEC CICS INVOKE SERVICE コマンドを呼び出します。 この EXEC CICS INVOKE SERVICE コマンドでパイプラインが呼び出されます。 パイプラインはアプリケーション言語構造を、サービス・プロバイダーが処理できる言語 (SOAP メッセージなど) へ変換します。

2. 要求をサービス・プロバイダーに送信する。

   CICS は、HTTP または WebSphere MQ のいずれかを使用して、リモート・サービス・プロバイダーへ要求メッセージを送信します。
3. サービス・プロバイダーから応答を受信する。

   サービス・プロバイダーの応答メッセージを受け取ると、CICS はそのメッセージをパイプラインに戻します。
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。

   パイプラインは、サービス・プロバイダーの応答メッセージをアプリケーション言語構造に変換します。これは、アプリケーション・プログラムに渡されます。その後、制御がアプリケーション・プログラムに戻されます。

パイプラインが適切に構成されていれば、パイプライン内の処理の一部を zEnterprise Application Assist Processor (zAAP) を使用して実行できます。詳しくは、『Java ベースの SOAP パイプライン』を参照してください。

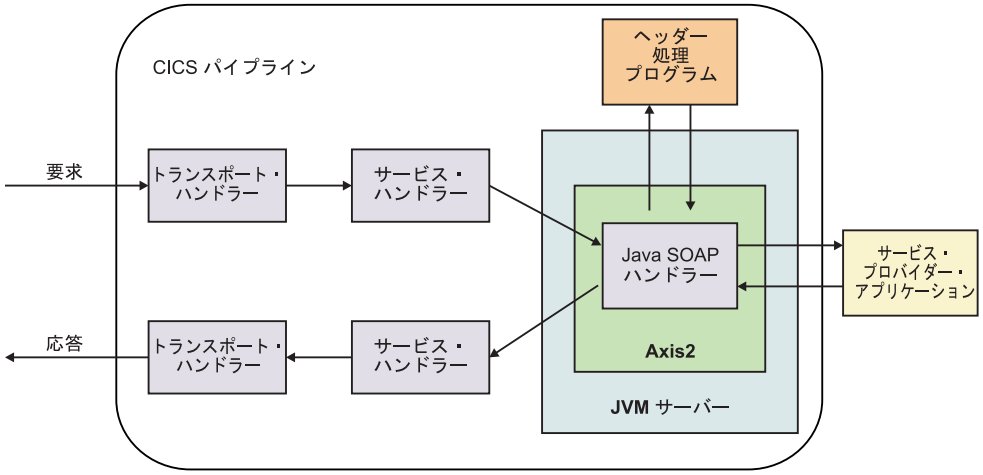
Java ベースの SOAP パイプライン

CICS は、Java ベースの Axis2 SOAP エンジンを使用してプロバイダーおよびリクエスター・パイプラインでの Web サービス要求を処理する機能をサポートしています。 Axis2 は Java を使用するため、そこでの SOAP 処理は zEnterprise Application Assist Processor (zAAP) へのオフロードに適格であるといえます。

Axis2 は、Apache 財団のオープン・ソース Web サービス・エンジンであり、Java 環境で SOAP メッセージを処理するために CICS で提供されています。 Java SOAP ハンドラーをパイプライン構成ファイルに追加し、Axis2 処理を扱うための JVM サーバーを作成するという方法で Axis2 を使用することもできます。

Axis2 を使用可能にする場合、パイプラインを使用する既存の Web サービスのバインディング・ファイルを再生成する必要はありません。 Axis2 を使用すると応答時間が遅くなる可能性がありますが、SOAP 処理の zAAP へのオフロードが可能です。 zAAP へのオフロードについて詳しくは、CICS における Java サポートを参照してください。

CICS がサービス・プロバイダーである場合、Java ベースの端末ハンドラーは、Axis2 を使用して要求メッセージの SOAP エンベロープを解析します。 SOAP メッセージに関連付けられている SOAP ヘッダーは、ヘッダー処理プログラムを使用して処理できます。 Axis2 は SOAP 応答メッセージの構成も行います。 この処理を次の図に示します。



CICS がサービス・リクエスターである場合、パイプラインの Java ベースの初期ハンドラーは、Axis2 を使用して要求メッセージの SOAP エンベロープを生成します。 SOAP メッセージに関連付けられている SOAP ヘッダーは、ヘッダー処理プログラムを使用して処理できます。 Axis2 は SOAP 応答メッセージの解析も行います。

Web サービス・アプリケーションおよび Java

プロバイダー・モードの SOAP パイプラインの場合、アプリケーション・ハンドラーを使用して、パイプラインの端末ハンドラーと Web サービス・アプリケーションの間で要求および応答メッセージが渡されます。アプリケーション・ハンドラーは、SOAP 要求の本体を処理して、アプリケーションがその要求を使用できるようにします。 さらに、アプリケーション・ハンドラーは、アプリケーションから返されたデータを使用して応答を生成します。 パイプラインの端末ハンドラーが Java ベースのメッセージ・ハンドラーである場合、提供されている DFHPITP アプリケーション・ハンドラーを指定するのではなく、提供されている Axis2 アプリケーション・ハンドラーをパイプライン構成ファイルで指定することができます。 そうすると、アプリケーション・ハンドラーの処理を zAAP にオフロードすることができます。 アプリケーション・ハンドラーについて詳しくは、 113 ページの『アプリケーション・ハンドラー』を参照してください。

リクエスター・モードの SOAP パイプラインの場合、Web サービス・アプリケーションは **EXEC CICS INVOKE SERVICE** コマンドを使用してパイプラインを呼び出します。そうすると、Web サービス・アプリケーションとパイプライン内の初期ハンドラーの間で要求メッセージおよび応答メッセージが渡されます。 Java ベースのハンドラーをパイプライン内の初期ハンドラーとして指定する場合、Axis2 によって **EXEC CICS INVOKE SERVICE** コマンドが処理されるため、この処理を zAAP にオフロードできます。 最初のハンドラーが Java ベースのハンドラーではない場合、**EXEC CICS INVOKE SERVICE** コマンドは CICS によって処理されます。

JVM サーバーでの Axis2 処理

Axis2 では JVM サーバーが必要です。これは、CICS で JVMSERVER リソースによって示されます。 JVM サーバーは、さまざまな Java プログラムから同時に行

われる複数の要求を単一の JVM で処理できるランタイム環境です。 JVM サーバーのクラスパスには、Axis2 Java アーカイブ・ファイルが含まれていなければなりません。 JVM プロファイルで JAVA_PIPELINE オプションを指定することにより、必要なすべての JAR ファイルを自動的にクラスパスに追加できます。 パイプライン構成ファイルでも、Axis2 をサポートするために構成されている JVMSERVER リソースを指していなければなりません。

JVM サーバーについて詳しくは、CICS における Java サポートを参照してください。

Axis2 ヘッダー・ハンドラー

既存のヘッダー処理プログラムを使用することもできますが、Java で Axis2 ハンドラーを作成し、SOAP ヘッダーを処理する方がより効果的です。 これらのハンドラーは JVM サーバーで実行することもできるので、オフロードに適格です。 Axis2 ハンドラーの作成についての詳細は、Writing Your Own Axis2 Module を参照してください。

ヘッダー・ハンドラー・プログラムは、Axis2 API を使用して、Axis2 環境、SOAP メッセージ、および個々の Web サービスを変更したり、それらと対話したりすることができます。 これらの API を使用して Axis2 をカスタマイズしないでください。CICS がエンジンを正しく稼働できないことを意味するように Axis2 が変更されてしまう可能性があります。 Axis2 ハンドラーは、CICS が Axis2 を使用する方法と互換性のある方法で、Axis2 環境と対話する場合にのみサポートされます。

Axis2 リポジトリー

Axis2 は、すべての構成ファイル、サービス、およびモジュールを保管するためにリポジトリーを使用します。 CICS では、z/OS UNIX 上の *usshome*/lib/pipeline/repository ディレクトリー (*usshome* は **USSHOME** システム初期設定パラメーターの値) にデフォルトのリポジトリーがあります。

このデフォルトのリポジトリーには、 Axis2 を使用するために CICS で必要な、構成ファイル axis2.xml が含まれています。 このファイルは、リポジトリー内の /conf サブディレクトリーにあります。 独自のリポジトリーを作成する場合、CICS が Axis2 と連動するように、このファイルをリポジトリーにコピーする必要があります。

ハンドラー・プログラムを登録していない場合、axis2.xml ファイルを編集しないでください。 このファイルは、CICS の内部パーツとして管理されているため、IBM サポートから要求されない限り、このファイルにその他のどのような変更も加えてはなりません。

Web サービスのデータ・フォーマット

異なる CICS テクノロジーを使って、同様に仕様に準拠するが物理的には異なる JSON データおよび XML データを生成できます。またそれらは、入力メッセージ内で検出されたエラーを異なる方法で報告することがあります。これは、データを検証するために検査を適用する順序によるものです。

CICS は、JSON データおよび XML データを自動的に変換するために、いくつかの異なるテクノロジーを使用します。これには、z/OS Connect for CICS (Java および非 Java バリアントの両方)、 Axis2、および PIPELINE の各リソースが含まれます。これらのテクノロジーは、関連仕様で規定された外部形式で JSON データおよび XML データを生成します。

同様に仕様に準拠するが物理的には異なるデータを表す方法は複数あります。 CICS テクノロジーは常に準拠するデータを生成しますが、それらの間には物理的な違いがある場合があります。たとえば、JSON 用の z/OS Connect for CICS オプションを Java と非 Java で切り替えると、生成される JSON に小さな相違点を検出することがあります。

そのような相違には、障害状態で報告されるエラー・メッセージや、空白文字の挿入方法、数値データの代替 (ただし等価) の表記、さらに特殊文字をエスケープする方法のバリエーションが含まれます。このような相違のさらなる変更は CICS に修理保守を適用した結果として、あるいは CICS の新しいリリースによって導入されることがあります。

JSON クライアントなどのパートナー・システムは、このような仕様に準拠するさまざまなバリエーションに対応するように作成されている必要があります。多くの場合、パートナーは JSON および XML との対話のために、広く使用されている業界共通のライブラリーやテクノロジーを活用します。これらのライブラリーは、そのようなフォーマットのわずかな違いを自動的に処理します。ただし、準拠性の低いパートナー・システムでは、さまざまな CICS テクノロジーのフォーマットの違いの検出や応答が異なることがあるため、標準ベースのパーサーを使用しないで JSON または XML を直接処理するアプリケーションを作成する場合には、注意が必要です。

JSON 要求のサービス・プロバイダーとしての CICS

外部 JSON クライアントにサービスを提供するために、CICS は要求を受け取り、パイプラインを介してそれをターゲット・アプリケーション・プログラムに渡す必要があります。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 JSON クライアントに戻されます。

CICS を JSON 要求のサービス・プロバイダーとして構成するには、次に示すいくつかの方法があります。

- z/OS Connect for CICS を使用する。

  z/OS Connect for CICS は、JSON を使用して CICS プログラムなどの z/OS アセットにアクセスするためのテクノロジーです。 z/OS Connect for z/OS の詳細、および z/OS Connect for CICS と CICS Java パイプラインの違いについては、z/OS Connect 入門を参照してください。
- CICS Java パイプラインを使用する。

  CICS Java パイプラインは、JSON を使用して CICS プログラムへのアクセスを可能にする、以前のバージョンの CICS で提供されたテクノロジーです。 CICS Java パイプラインについて詳しくは、Java ベースの SOAP パイプラインを参照してください。

- CICS Liberty JVM サーバーの JAX-RS 機能および JSON 機能を直接使用する。
- CICS 提供の端末ハンドラー DFHPIJT を使用する。

  プロバイダー・パイプラインに端末ハンドラー DFHPIJT を構成できます。こうすることで、JVM サーバーをインストールする必要なく、パイプラインは JSON 要求を処理できます。次の制限が適用されます。

  制約事項:
  - RESTful JSON Web サービスはサポートされていません。
  - パイプライン内のコンテキスト・スイッチはサポートされていません。
  - JSON パイプラインで SOAP および JSON Web サービスを使用することはできません。 DFHPIJT が扱うのは JSON メッセージのみです。 SOAP メッセージを受信すると、エラー応答になります。

CICS は JSON データを受信して、それを CICS アプリケーション・プログラムが理解できる構造化されたアプリケーション・データに変換します。 CICS アプリケーションからの応答は、アウトバウンド応答の JSON ペイロードに変換されます。変換を行うには、メッセージの構文解析が必要です。 z/OS Connect for CICS を使用する場合、Java ベースの JSON パーサーを使用するか、非 Java の同等のものを使用するオプションがあります。構成オプションは、z/OS Connect for CICS パイプライン構成ファイルで指定されています。 z/OS Connect for CICS を構成する方法について詳しくは、z/OS Connect for CICS の構成を参照してください。CICS Java パイプラインを使用する場合、構文解析は JVM サーバー内の Java を使用してのみ行われます。異なる構成を選ぶことによって、次のような違いがあります。

- Java を使用して JSON を構文解析した場合、この処理は zEnterprise Application Assist Processor (zAAP) によるオフロードに適格です (使用可能な場合)。処理のオフロードは、費用便益が期待できます。
- z/OS Connect for CICS の非 Java パーサーを使用する場合、一部のワークロードにはパフォーマンスまたはスループットの利点があります。益を受けるワークロードについて詳しくは、このリリースで利用可能なパフォーマンスに関する資料を参照してください。非 Java パーサーを使用した場合でも、多くの z/OS Connect for CICS インフラストラクチャー処理は zAAP へのオフロードに適格です。
- CICS 提供の端末ハンドラー DFHPIJT を使用すると、ワークロードによっては、パフォーマンスおよびスループットの面で利点が得られる可能性があります。この方法で JSON 要求を処理する場合、どの処理も zAAP へのオフロードの対象になりません。

z/OS Connect for CICS を使用した、**JSON** 要求のサービス・プロバイダーとしての **CICS:**

CICS が外部 JSON クライアントにサービスを提供するには、CICS が z/OS Connect for CICS に要求を受け取り、JSON メッセージを変換し、それをターゲット・アプリケーション・プログラムに渡す必要があります。アプリケーションからの応答は、同じメカニズムを介して JSON クライアントに戻されます。

図 17 は、CICS が z/OS Connect for CICS を使用するサービス・プロバイダーである場合に、外部 JSON クライアントからの要求を処理するために必要なアーキテクチャーおよびリソースの構成例を示しています。

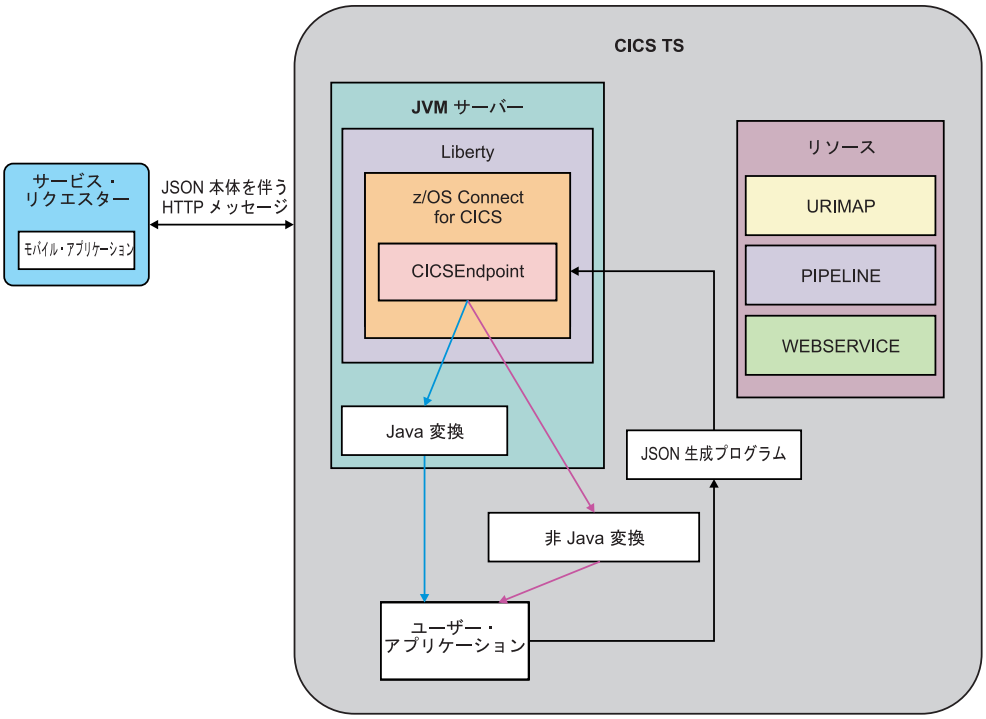


図 *17. z/OS Connect for CICS* を使用する *JSON* サービス・プロバイダーのアーキテクチャーおよびリソース

z/OS Connect for CICS を使用した JSON 要求の処理

要求を処理するため、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。

   WebSphere Liberty は、要求を受け取り、それを z/OS Connect for CICS に渡します。

2. USAGE 属性を JVMSERVER に設定して URIMAP 定義をスキャンすることで、CICS は要求の対象の URIMAP リソースを解決します。 URIMAP は、使用される WEBSERVICE リソースを識別します。 URIMAP からのトランザクション ID を使用して要求を処理する、新しい CICS タスクが開始されます。デフォルトでは、CPIH トランザクションが使用されます。

   WEBSERVICE リソースは、CICS が実行する処理を制御します。具体的には、WEBSERVICE リソースによって示される WSBind ファイルが、JSON と構造化されたアプリケーション・データとの間の変換に使用されます。 JSON Web サービスの WSBind ファイルは、ユーティリティー DFHLS2JS および DFHJS2LS を使用して生成されます。

注**:** スキーマに照らした JSON データの実行時検証はサポートされていません。JSON ペイロードで使用される WEBSERVICE リソースの VALIDATION 属性値は無視されます。
制約事項について詳しくは、『JSON Web サービスの制約事項』を参照してください。

3. z/OS Connect for CICS が、その構成内容に従って要求を処理します。 z/OS Connect for CICS が、グローバル・インターセプターを使用するように構成されている場合、この処理中にインターセプターが実行されます。
4. CICSEndpoint が制御を受け取ります。 JSON ペイロードは、パイプライン構成ファイルで指定された構成オプションに従って、構造化されたアプリケーション・データに変換されます。変換を実行する方法には 2 つのオプションがあります。
   - Java 変換が、JVM サーバー内で実行されます。この処理は、zAAP プロセッサーへのオフロードに適格です (使用可能な場合)。
   - 非 Java 変換が JVM サーバー外で実行されます。これは、一部のワークロードにパフォーマンスおよびスループットの利点をもたらす場合があります。益を受けるワークロードについて詳しくは、このリリースで利用可能なパフォーマンスに関する資料を参照してください。

   このマッピングは、WSBind ファイルの情報に従って実行されます。変換の出力は、いずれのケースのものも同等です。
5. CICS はアプリケーション・プログラムにリンクし、変換データを渡します。プログラムはこの入力を処理し、JSON 生成プログラムに応答を返します。
6. JSON 生成プログラムは、手順 5 の出力を使用して JSON 応答メッセージを生成します。このメッセージは、z/OS Connect for CICS を介してサービス・リクエスターに戻されます。

z/OS Connect for CICS は、次のような標準の RESTful メソッドをサポートする RESTful インターフェースも提供します。

POST

PUT

GET

DELETE

HEAD

このインターフェースは、通常の JSON 要求と同じように、パイプラインの構成された変換と生成プログラムを使用します。 RESTful JSON Web サービスについて詳しくは、RESTful JSON Web サービスの概念を参照してください。

多くの z/OS Connect for CICS インフラストラクチャー処理は、zEnterprise Application Assist Processor (zAAP) へのオフロードに適格です。

CICS Java パイプラインを使用した、**JSON** 要求のサービス・プロバイダーとしての **CICS:**

外部 JSON クライアントにサービスを提供するために、CICS は要求を受け取り、パイプラインを介してそれをターゲット・アプリケーション・プログラムに渡す必

要があります。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 JSON クライアントに戻されます。JSON 変換は、JVM サーバー内の Java を使用して実行されます。

図 18 は、CICS が Java パイプラインを使用するサービス・プロバイダーである場合に、外部 JSON クライアントからの要求を処理するために必要なアーキテクチャーおよびリソースの構成例を示しています。 JSON 要求に関するパイプライン処理は、Java パイプラインで SOAP 要求が CICS によって処理される方法と似ています。詳しくは、Java ベースの SOAP パイプラインを参照してください。

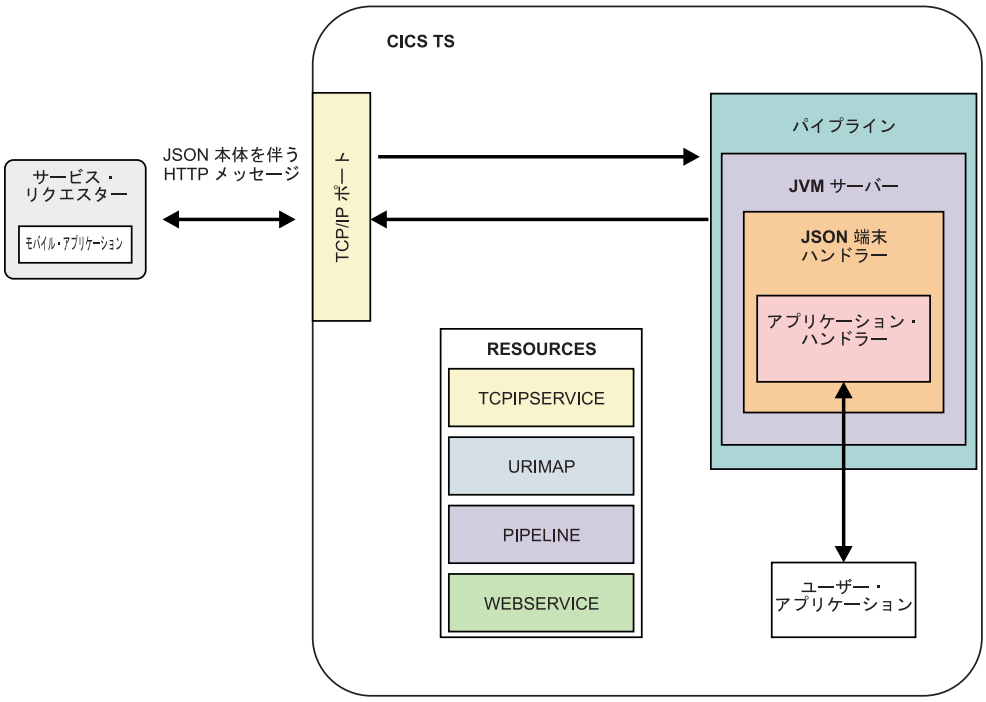


図 *18. CICS Java* パイプラインを使用する *JSON* サービス・プロバイダーのアーキテクチャーおよびリソース

JSON 要求の処理

要求を処理するため、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。

   TCPIPSERVICE リソースは着信要求のポートを指定します。 このポートは、CICS 提供のソケット・リスナー・タスク (CSOL) によってモニターされます。
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。

   適切なポートで要求メッセージを受け取ると、URIMAP リソース定義がスキャンされ、その中で USAGE 属性の設定が PIPELINE になっており、かつ PATH 属性の設定が要求で検出された URI になっている URIMAP 定義が検

出されます。 適切な URIMAP 定義が見つかった場合、その URIMAP 定義の PIPELINE および WEBSERVICE の各属性で指定された PIPELINE および WEBSERVICE 定義が使用されます。 URIMAP 定義の TRANSACTION 属性によって、パイプラインを処理するために接続されているトランザクションの名前が判別されます。 デフォルトでは、CPIH トランザクションが使用されます。 URIMAP 定義では、使用する PIPELINE および WEBSERVICE リソースも識別されます。

これらの PIPELINE および WEBSERVICE リソースは、その CICS が実行する処理を制御します。 特に WEBSERVICE リソースによって示される WSBind ファイルは、JSON と言語構造間のデータ変換に使用されます。JSON Web サービスの WSBind ファイルは、ユーティリティー DFHLS2JS および DFHJS2LS を使用して生成されます。

注**:** スキーマに照らした JSON データの実行時検証はサポートされていません。JSON ペイロードで使用される WEBSERVICE リソースの VALIDATION 属性値は無視されます。
制約事項について詳しくは、『JSON Web サービスの制約事項』を参照してください。

3. パイプライン処理が開始し、定義されているハンドラーを要求が流れていきます。現在 CICS が SOAP Web サービス用に提供しているハンドラーのいずれも JSON Web サービスに使用することは想定されていません。
4. パイプラインの終了時に、JSON ターミナル・ハンドラーが呼び出されます。このターミナル・ハンドラーは、Axis2 パイプラインとインターフェースで接続する Java プログラムです。ターミナル・ハンドラーは必要な Axis2 構成のセットアップを行い、HTTP 要求本体を使用して Axis2 エンジンを開始します。Axis2 パイプラインでは、JSON 本体 (存在する場合) が構文解析され、コンテンツを表す Java オブジェクト・モデルが構成されます。その後、CICS はアプリケーション・ハンドラーを呼び出します。アプリケーション・ハンドラーの主な役割は、要求の Java オブジェクト・モデル表現をアプリケーション・データにマッピングすることです。このマッピングは、WSBind ファイルの言語構造の記述を使用して実行されます。
5. 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。

   次に、アプリケーション・ハンドラーがアプリケーション・プログラムにリンクします。このプログラムはこの入力を処理し、アプリケーション・ハンドラーに応答を返します。
6. アプリケーション・プログラムによって戻されるデータを使用して応答を作成し、応答をサービス・リクエスターに送信する。

   アプリケーション・ハンドラーおよびメッセージ・ハンドラーは、サービス・プロバイダー・アプリケーションから受け取った応答メッセージを、元の要求の形式のメッセージに変換します。 このメッセージはサービス・リクエスターに戻されます。

パイプライン内の処理の一部を zEnterprise Application Assist Processor (zAAP) にオフロードできます。

SOAP 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

SOAP 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、パイプライン構成ファイルを作成し、多数の CICS リソースを作成する必要があります。

始める前に

Java パイプラインを使用する場合、JVM プロファイルで JAVA_PIPELINE=YES オプションが指定されている、JVMSERVER リソースが存在することを確認します。

JVM サーバーは、多くの Java パイプラインにおいて SOAP 処理を扱えます。

このタスクについて

CICS または CICSPlex® SM 機能を使用してローカル CICS 領域に PIPELINE リソースを定義できます。あるいは CICS Explorer® を使用して、ローカル CICS 領域または CICS バンドルに PIPELINE リソースを定義することもできます。 CICS Explorer を使って CICS バンドル内で PIPELINE リソースを定義するときには、パイプライン構成ファイルの作成も行い、それを CICS バンドル内にパッケージ化します。これにより、このファイルを別個に管理する必要がなくなります。また、PROGRAM リソースと WEBSERVICE リソースを CICS バンドル内に定義することもできます。 CICS バンドル内で WEBSERVICE リソースを定義するときには、Web サービス・バインディング・ファイルと WSDL 文書または WSDL アーカイブ・ファイルをインポートして、これらをバンドルに含めることができます。また、Web サービスをサポートするための URIMAP 定義を作成して、これらをバンドル内にパッケージ化することもできます。 CICS Explorer を使用して CICS バンドル内のリソースを作成および編集する方法について詳しくは、CICS Explorer 製品資料内の『Working with bundles』を参照してください。

手順

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。
   a. WebSphere 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。
      1) 詳しくは、WebSphere MQ トランスポートを使用するための CICS の構成を参照してください。
   b. 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を定義する TCPIPSERVICE リソースを定義する必要があります。
      1) 詳しくは、Web サービスのための CICS リソースを参照してください。
2. オプション: 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。

3. インバウンド Web サービス要求とその応答を処理するために、パイプライン構成ファイルに含めるメッセージ・ハンドラーおよびヘッダー処理プログラムを定義します。 CICS には次のハンドラーとヘッダー処理プログラムがあります。
   a. SOAP メッセージ・ハンドラー。SOAP 1.1 または 1.2 のメッセージを処理します。サービス・プロバイダー・パイプラインでは、1 つのレベルの SOAP だけをサポートできます。
   b. MTOM ハンドラー。MTOM/XOP 仕様に準拠する MIME Multipart/Related メッセージを処理します。
   c. Web サービスを保護するためのサポート。セキュア Web サービス・メッセージを処理します。
   d. Web サービス・トランザクションのサポート。アトミック・トランザクション・メッセージを処理します。
4. オプション: パイプラインで独自の処理を実行したい場合は、メッセージ・ハンドラーまたはヘッダー処理プログラムを作成する必要があります。 詳しくは、メッセージ・ハンドラーを参照してください。 カスタムのメッセージ・ハンドラー・プログラムを作成することにした場合は、パフォーマンスを最適化するには、プログラムをスレッド・セーフにする必要があります。
5. メッセージ・ハンドラー、ヘッダー処理プログラム、およびアプリケーション・ハンドラーを備えた XML パイプライン構成ファイルを作成します。
   a. CICS には、2 つの基本的なプロバイダー・モードのパイプライン構成ファイルのサンプル (`basicsoap11provider.xml` と `basicsoap11javaprovider.xml`) が用意されています。
   b. これらのサンプルを編集したり、必要に応じてさらにメッセージ・ハンドラーを追加したりできます。 サンプルは、ライブラリー `/usr/lpp/cicsts/cicsts55/samples/pipelines` (`/usr/lpp/cicsts/cicsts55` は z/OS UNIX 上の CICS ファイルのデフォルト・インストール・ディレクトリー) にあります。
   c. パイプライン構成ファイルで使用可能なオプションについて詳しくは、パイプライン構成ファイルを参照してください。
6. パイプライン構成ファイルを z/OS UNIX 内の適切なディレクトリーにコピーします。
7. パイプライン構成ファイルの権限を変更して、CICS 領域でファイルを読み取ることができるようにします。
8. 必要な個々のパイプライン構成ごとに、ステップ 5 から 7 を繰り返します。
9. PIPELINE リソースを作成します。
   a. PIPELINE リソースは、パイプライン構成ファイルの場所を定義します。また、Web サービス・バインディング・ファイルと WSDL (オプション) が格納される z/OS UNIX ディレクトリーである、ピックアップ・ディレクトリー を指定します。
   b. 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。
   a. PIPELINE リソースを作成すると、CICS は、指定したピックアップ・ディレクトリーに格納されているファイルを読み取り、WEBSERVICE リソースと URIMAP リソースを動的に作成します。

10. 自動インストール PROGRAM 定義を使用する場合を除き、パイプラインで実行するプログラムごとに、PROGRAM リソースを作成します。 このようなプログラムには、通常はトランザクション CPIH の下で実行するターゲット・アプリケーション・プログラムがあります。 トランザクションは、属性 TASKDATALOC(ANY) で定義されます。したがって、プログラムをリンク・エディットする際は、AMODE(31) オプションを指定する必要があります。

タスクの結果

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。

次のタスク

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か、または追加のパイプラインを作成する必要がある場合は、構成を拡張できます。

SOAP 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

SOAP 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、パイプライン構成ファイルを作成し、多数の CICS リソースを作成する必要があります。

始める前に

Java パイプラインを使用する場合、JVM プロファイルで JAVA_PIPELINE=YES オプションが指定されている、JVMSERVER リソースが存在することを確認します。JVMSERVER リソースを参照してください。

JVM サーバーは、多くの Java パイプラインにおいて SOAP 処理を扱えます。

このタスクについて

CICS または CICSPlex SM 機能を使用してローカル CICS 領域に PIPELINE リソースを定義できます。あるいは CICS Explorer を使用して、ローカル CICS 領域または CICS バンドルに PIPELINE リソースを定義することもできます。 CICS Explorer を使って CICS バンドル内で PIPELINE リソースを定義するときには、パイプライン構成ファイルの作成も行い、それを CICS バンドル内にパッケージ化します。これにより、このファイルを別個に管理する必要がなくなります。また、PROGRAM、WEBSERVICE、および URIMAP リソースを CICS バンドル内に定義することもできます。 CICS バンドル内で WEBSERVICE リソースを定義するときには、Web サービス・バインディング・ファイルと WSDL 文書または WSDL アーカイブ・ファイルをインポートして、これらをバンドル内にパッケージ化することができます。サービス・プロバイダー用に PROGRAM 定義を含めるよう選択できます。また、Web サービスをサポートするための URIMAP 定義を作成して、これらをバンドル内にパッケージ化することもできます。 CICS Explorer を使用して CICS バンドル内のリソースを作成および編集する方法について詳しくは、CICS Explorer 製品資料内の『Working with bundles』を参照してください。

手順

1. インバウンド Web サービス要求とその応答を処理するために、パイプライン構成ファイルに含めるメッセージ・ハンドラーおよびヘッダー処理プログラムを定義します。 CICS には次のハンドラーとヘッダー処理プログラムがあります。
   a. SOAP メッセージ・ハンドラー。SOAP 1.1 または 1.2 のメッセージを処理します。 サービス・リクエスター・パイプラインの 1 つのレベルの SOAP だけサポートできます。
   b. MTOM ハンドラー。MTOM/XOP 仕様に準拠する MIME Multipart/Related メッセージを処理します。
   c. セキュリティー・ハンドラー。セキュア Web サービス・メッセージを処理します。
   d. WS-AT ヘッダー処理プログラム。アトミック・トランザクション・メッセージを処理します。
2. オプション: パイプラインで独自の処理を実行したい場合は、メッセージ・ハンドラーまたはヘッダー処理プログラムを作成する必要があります。詳しくは、161 ページの『メッセージ・ハンドラー』を参照してください。カスタムのメッセージ・ハンドラー・プログラムを作成することにした場合は、パフォーマンスを最適化するには、プログラムをスレッド・セーフにする必要があります。
3. メッセージ・ハンドラーおよびヘッダー処理プログラムを備えた XML パイプライン構成ファイルを作成します。 CICS では、`basicsoap11provider.xml` と `basicsoap11javaprovider.xml` という 2 つの基本的なリクエスター・モードのパイプライン構成ファイルのサンプルが提供されています。必要に応じて、これをコピーしたり編集したりできます。 これらのサンプルは、ライブラリー `/usr/lpp/cicsts/cicsts55/samples/pipelines` (`/usr/lpp/cicsts/cicsts55` は z/OS UNIX 上の CICS ファイルのデフォルト・インストール・ディレクトリー) にあります。 パイプライン構成ファイルで使用可能なオプションについて詳しくは、101 ページの『パイプライン構成ファイル』を参照してください。
4. パイプライン構成ファイルを z/OS UNIX 内の適切なディレクトリーにコピーします。
5. パイプライン構成ファイルの権限を変更して、CICS 領域でファイルを読み取ることができるようにします。
6. 必要な個々のパイプライン構成ごとに、ステップ 3 から 5 を繰り返します。
7. PIPELINE リソースを作成します。PIPELINE リソースを参照してください。 PIPELINE リソースは、パイプライン構成ファイルの場所を定義します。 また、Web サービス・バインディング・ファイルと WSDL (オプション) が格納される z/OS UNIX ディレクトリーである、ピックアップ・ディレクトリー を指定します。 タイムアウトを秒単位で指定することもできます。これは、CICS が Web サービス・プロバイダーからの応答を待機する期間です。 パイプライン構成ファイルごとにこのステップを繰り返します。 PIPELINE リソースを作成すると、CICS は、指定したピックアップ・ディレクトリーに格納されているファイルを読み取り、WEBSERVICE リソースを動的に作成します (WEBSERVICE リソースを参照してください)。
8. 自動インストール PROGRAM 定義を使用する場合を除き、パイプラインで実行するプログラムごとに、PROGRAM リソースを作成します。PROGRAM リソースを参照してください。 このようなプログラムには、通常はトランザクシ

ョン CPIH の下で実行するサービス・リクエスター・アプリケーション・プログラムがあります。トランザクションは、属性 TASKDATALOC(ANY) で定義されます。したがって、プログラムをリンク・エディットする際は、AMODE(31) オプションを指定する必要があります。

9. オプション: CICS の URIMAP リソースを HTTP クライアントとして作成の指示に従って、要求を行うためにサービス・リクエスターが使用する各 URI へのクライアント要求のために、URIMAP リソース (URIMAP リソースを参照) を作成します。 この URI は、URIMAP リソースを使用する代わりに、プログラム内の **INVOKE SERVICE** コマンドで直接指定できます。ただし、URIMAP リソースを使用した場合は、サービス・プロバイダーの URI が変更されてもアプリケーションを再コンパイルする必要がありません。 URIMAP リソースによって、接続プールを実装することもできます。接続プールでは、CICS は使用後のクライアント接続をオープンしたままにして、同じアプリケーションが以降の要求で再利用したり、同じサービスを呼び出す別のアプリケーションが再利用したりできます。

タスクの結果

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。

次のタスク

追加のパイプラインを作成する必要がある場合は、構成を拡張できます。

JSON 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

JSON 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、パイプライン構成ファイルを作成し、多数の CICS リソースを作成する必要があります。

始める前に

CICS で JSON サービスを実装するためには、いくつかの異なるテクノロジーがあります。最も機能が豊富なオプションは z/OS Connect です。このタスクでは代替オプションについて説明します。

z/OS Connect を使用しない場合は、CICS を JSON 要求のサービス・プロバイダーとして使用するか、またはリンク可能なインターフェースを使用して JSON を変換します。これらの方式を使用するには、**JAVA_PIPELINE=YES** オプションを指定する JVM プロファイルを使用して JVMSERVER リソースを定義し、インストールします。 DFHAXIS という名前のサンプル JVMSERVER リソース定義がグループ DFH$AXIS の中にあります。

注: ここで説明するインフラストラクチャーでは、JSON サービス・プロバイダーへの接続に z/OS Connect for CICS を使用せずに、JVM サーバー内の Java 解析を使用して JSON メッセージを解析することを想定しています。 Java 以外の JSON 解析を使用する場合は、z/OS Connect for CICS を使用して JSON Web サービス

に接続する必要があります。 z/OS Connect for CICS のセットアップについて詳しくは、z/OS Connect for CICS の構成を参照してください。

手順

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。

   インバウンド要求を受信するポートを定義する TCPIPSERVICE リソースを定義します。詳しくは、『Web サービスのための CICS リソース』を参照してください。

2. インバウンド Web サービス要求とその応答を処理するために、パイプライン構成ファイルに含めるメッセージ・ハンドラーを定義します。

   パイプラインで独自の処理を実行したい場合は、メッセージ・ハンドラーを作成する必要があります。詳しくは、『メッセージ・ハンドラー』を参照してください。カスタムのメッセージ・ハンドラー・プログラムを作成することにした場合は、パフォーマンスを最適化するには、プログラムをスレッド・セーフにする必要があります。

3. メッセージ・ハンドラー、ヘッダー処理プログラム、およびアプリケーション・ハンドラーを備えた XML パイプライン構成ファイルを作成します。 CICS には、基本的なプロバイダー・モードのパイプライン構成ファイルのサンプル `jsonjavaprovider.xml` が用意されています。このサンプルを編集して、必要に応じてさらにメッセージ・ハンドラーを追加できます。このサンプルは `/usr/lpp/cicsts/cicsts55/samples/pipelines` ディレクトリーの中にあります (`/usr/lpp/cicsts/cicsts55` は z/OS UNIX 上の CICS ファイルのデフォルト・インストール・ディレクトリー)。パイプライン構成ファイルで使用可能なオプションについて詳しくは、 119 ページの『サービス・プロバイダーおよびサービス・リクエスターのパイプラインで使用されるエレメント』を参照してください。

4. パイプライン構成ファイルを z/OS UNIX 内の適切なディレクトリーにコピーします。

5. パイプライン構成ファイルの権限を変更して、CICS 領域でファイルを読み取ることができるようにします。

6. PIPELINE リソースを作成します。 PIPELINE リソースは、パイプライン構成ファイルの場所を定義します。 このリソースはさらに、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は、Web サービス・バインディング・ファイルが保管される z/OS UNIX ディレクトリーです。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。 PIPELINE リソースをインストールするか PIPELINE SCAN を実行する場合、CICS は指定したピックアップ・ディレクトリーにある `.wsbind` ファイルを読み取り、適切な WEBSERVICE リソースと URIMAP リソースを動的に作成します。

7. 自動インストール PROGRAM 定義を使用する場合を除き、パイプラインで実行するプログラムごとに、PROGRAM リソースを作成します。 このようなプログラムには、通常はトランザクション CPIH の下で実行するターゲット・アプリケーション・プログラムがあります。 トランザクションは、属性 `TASKDATALOC(ANY)` で定義されます。したがって、プログラムをリンク・エディットする際は、`AMODE(31)` オプションを指定する必要があります。

タスクの結果

それぞれのサービス・プロバイダーに必要なインフラストラクチャーを作成したので、これらのリソースを CICS システム上にインストールできるようになりました。

次のタスク

リソースをインストールします。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か、または追加のパイプラインを作成する必要がある場合は、構成を拡張できます。

非 Java JSON 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

JSON 要求を処理するための非 Java 環境をセットアップできます。非 Java JSON 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、パイプライン構成ファイルを作成し、いくつかの CICS リソースを作成する必要があります。

手順

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。

   インバウンド要求を受信するポートを定義する TCPIPSERVICE リソースを定義します。詳しくは、『Web サービスのための CICS リソース』を参照してください。

2. インバウンド Web サービス要求とその応答を処理するために、パイプライン構成ファイルに含めるメッセージ・ハンドラーを定義します。

   パイプラインで独自の処理を実行したい場合は、メッセージ・ハンドラーを作成する必要があります。詳しくは、『メッセージ・ハンドラー』を参照してください。カスタムのメッセージ・ハンドラー・プログラムを作成することにした場合は、パフォーマンスを最適化するには、プログラムをスレッド・セーフにする必要があります。

3. メッセージ・ハンドラーを含む XML パイプライン構成ファイルを作成します。

   構成ファイルの <terminal_handler> エレメントに、図 19 に示すように端末ハンドラー・プログラム DFHPIJT を指定する必要があります。DFHPIJT は CICS 提供の JSON ハンドラー・プログラムであり、JSON メッセージの非 Java 処理を可能にします。

```
<service>
  <terminal_handler>
    <handler>
        <program>DFHPIJT</program><handler_parameter_list/>
    </handler>
  </terminal_handler>
</service>
```

図 *19. JSON* メッセージの非 *Java* 処理のための端末ハンドラー *DFHPIJT* の指定

注: DFHPIJT を端末ハンドラーとして使用する場合は、パイプライン構成ファイルにアプリケーション・ハンドラーを定義しないでください。つまり、パイプライン構成ファイルに <apphandler> エレメントが含まれていてはなりません。アプリケーション・ハンドラーを指定しても、呼び出されません。

パイプライン構成ファイルで使用可能なオプションについて詳しくは、 119 ページの『サービス・プロバイダーおよびサービス・リクエスターのパイプラインで使用されるエレメント』を参照してください。

4. パイプライン構成ファイルを z/OS UNIX 内の適切なディレクトリーにコピーします。
5. パイプライン構成ファイルの権限を変更して、CICS 領域でファイルを読み取ることができるようにします。
6. PIPELINE リソースを作成します。 PIPELINE リソースは、パイプライン構成ファイルの場所を定義します。 このリソースはさらに、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は、Web サービス・バインディング・ファイルが保管される z/OS UNIX ディレクトリーです。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。 PIPELINE リソースをインストールするか PIPELINE SCAN を実行する場合、CICS は指定したピックアップ・ディレクトリーにある .wsbind ファイルを読み取り、適切な WEBSERVICE リソースと URIMAP リソースを動的に作成します。
7. 自動インストール PROGRAM 定義を使用する場合を除き、パイプラインで実行するプログラムごとに、PROGRAM リソースを作成します。 このようなプログラムには、通常はトランザクション CPIH の下で実行するターゲット・アプリケーション・プログラムがあります。 トランザクションは、属性 TASKDATALOC(ANY) で定義されます。したがって、プログラムをリンク・エディットする際は、AMODE(31) オプションを指定する必要があります。

タスクの結果

それぞれのサービス・プロバイダーに必要なインフラストラクチャーを作成したので、これらのリソースを CICS システム上にインストールできるようになりました。

次のタスク

リソースをインストールします。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か、または追加のパイプラインを作成する必要がある場合は、構成を拡張できます。

z/OS Connect for CICS の構成

CICS での z/OS Connect の構成

z/OS Connect は、CICS の JSON サービスおよび API を実装するための IBM 最高のテクノロジーです。

z/OS Connect はスタンドアロンで構成することも、CICS の Liberty JVM サーバーでホストすることもできます。スタンドアロン構成について詳しくは、z/OS Connect Enterprise Edition V3.0 の製品資料を参照してください。

ここでは、CICS アドレス・スペースで z/OS Connect を構成する方法について説明します。

z/OS Connect および z/OS Connect for CICS 1.0 という 2 つのメジャー・バージョンがあり、さらに高度な z/OS Connect Enterprise Edition バージョンもあります。

要件に一致するタスクを選択してください。

z/OS Connect for CICS 1.0の構成:

z/OS Connect for CICS 1.0 は、CICS Transaction Server の一部として配布されています。JSON サービスをデプロイするためには、z/OS Connect 用の JVM サーバーを構成し、パイプライン構成およびリソースをセットアップしておく必要があります。この初期構成は 1 回限りのアクティビティーです。

始める前に

CICS に WebSphere Liberty JVM サーバーが既に構成されていますか? z/OS Connect とその他の関係ないサービスを同じ WebSphere Liberty 環境でホストすることは可能ですが、z/OS Connect 専用の JVM サーバーを別途構成することをお勧めします。

z/OS Connect for CICS 1.0 を専用の CICS 領域/CICS 領域グループでホストし、さらに分散プログラム・リンクのメカニズムを使用して、アプリケーションが所有する CICS領域で CICS プログラムを呼び出すことができます。

手順

1. JVMSERVER を作成し、WebSphere Liberty をサポートするように構成します。WebSphere Liberty JVMSERVER の作成方法について詳しくは、Liberty JVM サーバーの構成を参照してください。
2. セキュリティー要件のために WebSphere Liberty を構成します。デフォルトで、WebSphere Liberty ではクライアント認証 SSL 証明書を使用する必要があります。この HTTP 基本認証を有効にするには、 `server.xml` ファイルに次の構成オプションを追加します。

   ```
   <!-- Allow fail-over to HTTP Basic Authentication -->
   <webAppSecurity allowFailOverToBasicAuth="true"/>
   ```

   z/OS Connect のユーザーに **`zosConnectAccess`** セキュリティー・ロールを付与することも必要です。 WebSphere Liberty セキュリティーについて詳しくは、Liberty JVM サーバーのセキュリティーの構成 (Configuring security for a Liberty JVM server)を参照してください。z/OS Connect セキュリティーについては、z/OS Connect のセキュリティーを参照してください。
3. WebSphere Liberty 環境用の `server.xml` ファイル内の `<featureManager>` リストを更新して `<feature>cicsts:zosConnect-1.0</feature>` 機能を含めます。次に例を示します。

   ```
   <featureManager>
       <feature>cicsts:core-1.0</feature>
       <feature>transportSecurity-1.0</feature>
       <feature>cicsts:zosConnect-1.0</feature>
   </featureManager>
   ```

4. 次のステートメントを server.xml ファイルに追加して、z/OS Connect for CICS 1.0 Service Controller を定義します。

```
<com.ibm.cics.wlp.zosconnect.CICSEndpoint
    id="com.ibm.cics.wlp.zosconnect.CICSEndpointService"/>
```

5. JVMSERVER をインストールします。生成された messages.log ファイルで エラー・メッセージあるいは警告メッセージがないか確認します。このログには、z/OS Connect for CICS 1.0 から戻されたメッセージを含め、WebSphere Liberty Server によって生成された次のようなメッセージが含まれます。

```
SRVE0169I: Web モジュールをロード中: z/OS Connect。
SRVE0250I: Web モジュール z/OS Connect は、default_host にバインドされています。
```

6. XML パイプライン構成ファイルを作成します。サンプルのパイプライン構成ファイル jsonzosconnectprovider.xml がディレクトリー/usr/lpp/cicsts/cicsts55/samples/pipelines/ (ここで /usr/lpp/cicsts/cicsts55 は z/OS UNIX の CICS ファイルのデフォルト・インストール・ディレクトリー) にあります。Liberty JVM サーバー内で Java を使用して JSON を解析するか (デフォルト)、Java 以外の JSON パーサーを使用するかを判断する必要があります。
   - Liberty JVM サーバー内で Java を使用して JSON を解析するには、サンプルのパイプライン構成ファイルを使用できます。その際、<jvmserver> 要素の DFHWLP をステップ 1 の JVMSERVER の名前に置き換えてください。
   - Java 以外のパーサーを使用して JSON を解析するには、サンプルの構成ファイルを変更して、以下の例のように java_parser="no" 属性を <provider_pipeline_json> 要素に追加します。

```
<?xml version="1.0" encoding="EBCDIC-CP-US"?>
<provider_pipeline_json java_parser="no"
    xmlns="http://www.ibm.com/software/htp/cics/pipeline">
        <jvmserver>DFHWLP</jvmserver>
</provider_pipeline_json>
```

   DFHWLP をこのプロシージャーの開始時に作成した JVMSERVER の名前と置き換えます。

7. パイプライン構成ファイルを zFS 内の適切なディレクトリーにコピーし、CICS 領域がファイルを読み取れるようなファイル権限になっていることを確認します。 詳細については、『パイプライン構成ファイル』を参照してください。
8. PIPELINE リソースを作成します。 PIPELINE リソースは、CONFIGFILE 属性にパイプライン構成ファイルの場所を定義します。
9. オプション: z/OS Connect 用のデフォルトの URIMAP リソースを作成します。 URIMAP リソースは、トランザクションおよびデフォルト・ユーザー ID を z/OS Connect の作業に関連付けるために使用されます。1 つ以上の URIMAP リソースを使用して、z/OS Connect のデフォルト・ポリシーを構成できます。 URIMAP の構成例と構成オプションの詳細については、z/OS Connect サービスおよび API の許可の構成を参照してください。

   注:

   z/OS Connect は、HTTP 要求ごとに別の認証を行います。その結果、CICS で実行されるアプリケーション・タスクは通常、URIMAP からの初期ユーザー

ID よりも限定的なユーザー ID に関連付けられます。初期ユーザー ID が有効なのは、z/OS Connect 内でユーザー固有の認証が行われるまでに限られます。

タスクの結果

z/OS Connect for CICS 1.0 インスタンスが構成されました。 URL `https://hostname:portnumber/zosConnect/services` を Web ブラウザーに入力して基本構成をテストできます。ここで、*hostname* は z/OS Connect for CICS 1.0 をホストしている CICS 領域が実行されているシステムの IP アドレスまたはホスト名で、*portnumber* は `server.xml` ファイルの `<httpEndpoint>` 要素で指定された **httpsPort** です。Web ブラウザーにはインストール済みのサービスのリストが表示されますが、インストールされているサービスがまだないため、リストは空です。

目的のサービス・リストではなく `HTTP 403 AuthorizationFailed` 応答を受け取った場合は、ステップ 2 のセキュリティー構成を確認してください。z/OS Connect を使用する権限が認証済みユーザーに付与されていない可能性があります。

次のタスク

これで、JSON Web サービスを z/OS Connect for CICS 1.0 にデプロイする準備ができました。

***CICS JSON Web** サービス用の **z/OS Connect for CICS** の構成:*

z/OS Connect for CICS を最初に構成した後に、それを CICS JSON Web サービス用に構成することができます。 JSON Web サービスは、他の CICS PIPELINE 環境にデプロイする方法と同様の方法で z/OS Connect for CICS にデプロイされます。

始める前に

このタスクを開始する前に、z/OS Connect の基本構成を完了する必要があります。z/OS Connect for CICS 1.0 については、 82 ページの『z/OS Connect for CICS 1.0の構成』を参照してください。z/OS Connect Enterprise Edition については、 89 ページの『z/OS Connect Enterprise Editionの構成』を参照してください。また、デプロイするサービスごとに JSON WSBind ファイルが必要です。このようなバインディング・ファイルは、CICS JSON アシスタント (**DFHLS2JS** および **DFHJS2LS** ユーティリティー・プログラム) を使用して生成できます。これらのユーティリティー・プログラムについて詳しくは、CICS JSON アシスタントを参照してください。

このタスクについて

サービスは z/OS Connectに、z/OS Connect 管理対象サービスとして、または CICS 管理対象サービスとしてデプロイできます。このトピックでは、これらのサービスを CICS 管理対象サービスとしてデプロイする方法について説明します。サービスを CICS 管理対象サービスとしてデプロイすることにより、サービスごとに WEBSERVICE リソースが存在するようになります。このデプロイメント・メカニズムは、CICS の古い JSON Web サービス・テクノロジーと互換性があります。古

いテクノロジーでは、サービスを z/OS Connect にデプロイするときに、SOAP Web サービスなどの他の CICS PIPELINE 環境と同様の方法でデプロイされます。

z/OS Connect Enterprise Edition V3 がある場合は、z/OS Connect Enterprise Edition V3.0 の製品資料で説明されているように、サービスを z/OS Connect 管理対象リソースとしてデプロイすることによって、より良い結果を得られる場合があります。

注: このタスクは、z/OS Connect Enterprise Edition V1.0、または CICS TS で提供される CICS サービス・プロバイダーを使用する場合に適用されます。 z/OS Connect Enterprise Edition v3.0 で提供される CICS サービス・プロバイダーを使用する場合は、z/OS Connect Enterprise Edition V3.0 製品資料内の『サービス・アーカイブ管理の自動化』を参照してください。

手順

- WSBind ファイルを適切な PIPELINE リソースに関連付ける WEBSERVICE リソースをインストールします。 WEBSERVICE には分かりやすい名前を選択してください。この名前は z/OS Connect でもサービスの名前として使用されます。必要に応じて、**PERFORM PIPELINE SCAN** コマンドを使用して WEBSERVICE リソースをインストールできます。どちらの方法を選択して使用するとしても、URIMAP に関する以下の情報を考慮するようにしてください。

  z/OS Connect 内のサービスが使用可能な URI は、次のいずれかの場所から取得されます。

  - z/OS Connect が使用するデフォルトの命名規則。
  - WSBind ファイルに保管されている URI (**PERFORM PIPELINE SCAN** コマンドを使用して関連 WEBSERVICE リソースがインストールされた場合)。
  - Liberty `server.xml` ファイル内のサービス固有の構成 (**invokeURI** 構成パラメーターが使用される場合)。
  - z/OS Connect Enterprise Edition のみ: サービスをカプセル化した API のアプリケーション・アーカイブ・ファイル (`.aar` ファイル)。

  デフォルトの命名規則を使用する場合、サービスは通常、次の URI で公開されます。

  ```
  https://<hostname>:<port>/zosConnect/services/<Service Name>?action=invoke
  ```

  この例では、`<Service Name>` はサービスの名前 (より具体的に言えば、WEBSERVICE リソースの名前) を表し、`<hostname>` および `<port>` は Liberty サーバーの構成から取得されます。

- オプション: z/OS Connect サービス用の URIMAP リソースをインストールします。 URIMAP リソースは、サービスの作業を CICS 内の特定のトランザクション ID と初期ユーザー ID に関連付けるために、CICS によって使用されます。 単一の URIMAP を使用して 1 つ以上のサービスを構成できます。z/OS Connect に対応する CICS を構成するときに、デフォルトの URIMAP リソースを定義することもできます。 z/OS Connect サービスの URI と一致する URIMAP が存在しなければ、アプリケーションのタスクはトランザクション CJSA のもとで実行され、CICS デフォルト・ユーザー ID (通常は CICSUSER)

が初期ユーザー ID として使用されます。初期ユーザー ID の役割について詳しくは、z/OS Connect サービスおよび API の許可の構成を参照してください。

URIMAP リソースを作成することを選択した場合、関連付けられたユーザー ID には指定されたトランザクションを実行する権限が必要です。

URIMAP を使用して URI を特定の WEBSERVICE リソースに関連付けることもできます。URIMAP が WEBSERVICE に密にバインドされている場合は、URIMAP によって突き合わされるすべての HTTP 要求にターゲット WEBSERVICE が使用されます。 WEBSERVICE が指定されない場合は、z/OS Connect サービスの名前に基づいて WEBSERVICE が選択されます。URIMAP の WEBSERVICE 属性がブランクのままの場合は、z/OS Connect 自らマッピングを行うので、z/OS Connect Enterprise Edition は API 内で異なるサービスにそれぞれ異なる HTTP メソッドを関連付けられるようになります。

- オプション: `server.xml` ファイルを更新します。 使用パターンによっては、Liberty サーバー構成ファイル `server.xml` の変更が必要な場合があります。通常、サービスのための特殊なケースの処理を必要としない限り、`server.xml` ファイルでサービス固有の変更を行う必要はありません。例えば、`server.xml` ファイルで z/OS Connect インターセプターの特定のセットをサービスに関連付けたり、z/OS Connect のデフォルトの命名規則と一致しない URI を使用してサービスを公開したりすることもできます。

  `server.xml` ファイルでサービスを定義するには、以下のステップを使用します。

  1. 目的とするサービス名が CICS 内の WEBSERVICE リソースの名前と一致しないことを確認します。CICS は構成情報を自動的に z/OS Connect に注入します。明示的に定義されたサービスと CICS WEBSERVICE が同じ名前の場合、結果の動作は予測不能です。
  2. `server.xml` ファイルの **`invokeURI`** 属性に、URIMAP で使用される URI と一致する適切な値を設定します。
  3. **`CICSEndpointService`** `serviceRef` エレメントにサービスをバインドします。
  4. サービスの URI を使用される厳密な WEBSERVICE に対応させる URIMAP が存在することを確認します。ステップ **2** で **`invokeURI`** 属性を設定した場合、URIMAP はその URI と一致しなければなりません。そうでなければ、URIMAP は z/OS Connect のデフォルトの URI 命名規則を前提としなければなりません。

  次の例は、`server.xml` ファイルでの明示的な z/OS Connect for CICS 1.0 サービス宣言を示しています。

```
<zosConnectService invokeURI="/json/myCustomService"
              serviceName="CICSService1"
              serviceRef="com.ibm.cics.wlp.zosconnect.CICSEndpointService"/>
```

  次の例は、z/OS Connect Enterprise Edition の場合の `server.xml` ファイルでの同等の宣言を示しています。

```
<zosconnect_zosConnectService invokeURI="/json/myCustomService"
              serviceName="CICSService1"
              serviceRef="com.ibm.cics.wlp.zosconnect.CICSEndpointService"/>
```

どちらの例も、"/json/myCustomService" URI が z/OS Connect for CICS 受信プログラム CICSService1 に関連付けられます。

タスクの結果

URL https://*hostname*:*portnumber*/zosConnect/services を Web ブラウザーに入力することによって、z/OS Connect for CICS の構成をテストできます。*hostname* は、z/OS Connect for CICS をホストする CICS 領域の IP アドレスまたは名前です。*portnumber* は、server.xml 構成ファイルの <httpEndpoint> セクションに構成されている **httpsPort** です。 Web ブラウザーには、インストール済みのサービスのリストが表示されます。

これで、同じ *hostname* と *portnumber* を使用する JSON クライアントからサービスを呼び出す準備ができました。

インストールされた各サービスの項目が zosConnect/services リストにあります。次に例を示します。

```
{
"id":"EXAMPLE",
"name":"EXAMPLE",
"url":"https://hostname:portnumber/zosConnect/services/EXAMPLE",
"protocol":"REST",
"description":"CICS Service"
}
```

この例では、関連付けられた WEBSERVICE リソースには "EXAMPLE" という名前が付けられており、このサービス定義は EXAMPLE WEBSERVICE リソースがインストールされたときに CICS によって動的に作成されています。 Web ブラウザーを使用して https://hostname:portnumber/zosConnect/services/EXAMPLE にアクセスできます。アクセスすると、サービスに関する詳細が記述された、次のような文書が返されます。

```
{
"id":"EXAMPLE",
"name":"EXAMPLE",
"protocol":"REST",
"description":"CICS Service",
"restEndpoints":[
      {
       "name":"EXAMPLE",
       "address":"hostname:portnumber/jsonTests/myExampleService"
      }
   ]
}
```

この例では、"address" はサービスが公開される URI です。この URI は、WSBind ファイル内の情報から、または **invokeURI** 属性から、あるいはデフォルトの z/OS Connect 命名規則から得られる可能性があります。

要求が Liberty JVM サーバーに到着すると、server.xml 構成ファイルの情報を使用する z/OS Connect for CICS 受信プログラムがその要求に関連付けられます。この処理を実行するために新しい CICS タスクが開始され、URIMAP の情報を使用する特定の WEBSERVICE リソースと関連付けられます。データ変換プロセスが Liberty JVM サーバー内で行われ、WSBind ファイルで指定されているターゲットの CICS プログラムが接続されます。

JSON Web サービスから ***z/OS Connect*** へのマイグレーション**:**

JSON サービスは、CICS の古い JSON Web サービス・テクノロジーから z/OS Connect に再デプロイできます。また、Mobile Extensions の古いフィーチャー・パックからも再デプロイできます。こうした再デプロイには、互換モードで z/OS Connect を構成する必要があります。このモードは z/OS Connect for CICS 1.0 でも z/OS Connect Enterprise Edition でも使用できます。

このタスクについて

このタスクで *z/OS Connect* とは、z/OS Connect for CICS 1.0 と z/OS Connect Enterprise Edition の両方を指します。

Java Pipelines for JSON (CICS Transaction Server Feature Pack for Mobile Extensions V1.0 で使用) に含まれる JSON Web サービスを z/OS Connect に再デプロイする方法が、プロセスとして簡単です。 Java Pipelines for JSONと z/OS Connect に使用される WSBind ファイルは、同じツール (DFHLS2JS および DFHJS2LS) を使用して生成され、相互に完全な互換性があります。CICS でこの機能を探索するなら、IBM Redbooks: Implementing IBM CICS JSON Web Services for Mobile Applications に JSON Web サービスの例があります。

z/OS Connect 環境では、クライアント・アプリケーションと CICS の間で SSL を使用することをお勧めします。既存の Java Pipelines for JSON環境で SSL を使用していない場合は、z/OS Connect への移行には追加の手順を必要とします。

手順

1. z/OS Connect for CICS の構成の手順を使用して、必要な z/OS Connect インフラストラクチャーを作成します。この構成の一環として、Liberty JVM サーバーが接続の着信を listen する SSL TCP/IP ポート番号を選択します。次の 2 つのオプションから検討します。
   a. Java Pipelines for JSONの TCPIPSERVICE によって使用されるポート番号とは異なるポート番号を選択します。このオプションは、両方の環境をそれぞれ異なる TCP/IP ポートに同時にインストールできるという利点があります。これは、新しい JSON サービスを対象とするようにクライアント・プログラムを更新する必要があることを意味します。以前の環境で SSL を使用していなかった場合は、z/OS Connect への移行で URI に変更を加える必要があるため、このオプションの方が適しています。
   b. Java Pipelines for JSONの TCPIPSERVICE で使用しているのと同じポート番号を選択します。クライアント・プログラムによって使用される URI でポート番号を変更する必要はありませんが、2 つの環境を同時にインストールすることはできません。 URI は、その他の理由により変更を加える必要がある場合があります。例えば、SSL を有効にするときの HTTP から HTTPS への切り替えです。
2. WSBind ファイルを z/OS Connect にデプロイします。既存の WSBind ファイルは、z/OS Connect と完全に互換性があります。 CICS JSON Web サービス用の z/OS Connect の構成に記載されている手順に従って、新しい JSON Web サービスを z/OS Connect にデプロイします。 CICS では、同じ名前の 2 つの WEBSERVICE リソースを両方インストールすることはできません。したがって、次のいずれかを行います。

a. 新しい WEBSERVICE を以前のものと同じ名前でインストールするために、元の WEBSERVICE を破棄します。
b. 競合を避けるために、新しい WEBSERVICE の名前を変更します。

3. PIPELINE ハンドラー・プログラムを使用して Java Pipelines for JSONの処理をカスタマイズした場合、そのカスタマイズが引き続き必要かどうか検討します。必要な場合は、z/OS Connect Interceptor プログラムを相当する関数で作成して、それらをグローバル・インターセプターとしてデプロイします。 z/OS Connect Interceptor について詳しくは、 z/OS Connect インターセプターの定義を参照してください。

タスクの結果

これで、新しい z/OS Connect JSON Web サービスをテストする準備ができました。Java Pipelines for JSONで使用したポート番号と同じポート番号を使用してサービスをデプロイした場合、クライアントに変更を加える必要はありません (ただし、SSL を有効にするなどのセキュリティー構成を変更した場合を除きます)。ポート番号またはサービスの URI を変更した場合は、クライアントに変更を加える必要があります。

z/OS Connect Enterprise Editionの構成**:**

z/OS Connect Enterprise Edition は個別注文の製品であり、CICS TS の一部として提供される製品ではありません。これは、z/OS Connect for CICS 1.0 の基本サービスを拡張して追加機能を提供し、CICS の古い JSON Web サービスベースのサービスをホストしたり、標準的な z/OS Connect デプロイメント成果物 (SAR ファイルや AAR ファイル) を使用したりすることができます。 JSON サービスをデプロイしたり PIPELINE リソースを構成したりするには、その前に JVMSERVER リソースを構成する必要があります。この初期構成は 1 回のみ必要です。

始める前に

z/OS Connect Enterprise Edition ランタイムをインストールします。 z/OS Connect Enterprise Edition V3.0 の製品資料の説明に従ってください。z/OS Connect Enterprise Edition を CICS に組み込んで実行する準備をする場合、別個の Liberty サーバー・インスタンスを作成する必要はありません。 zFS でファイル・システム・コンポーネントを使用できることを確認する必要があります。z/OS Connect Enterprise Edition で提供される API Toolkit のインストールは、このタスクの一部ではありません。

このタスクについて

以前に z/OS Connect for CICS 1.0 をインストールした場合は、適切な構成が既に存在している可能性があり、そうであれば以下のタスクでいくつかのステップを省略できます。省略できるステップは、手順の中で示しています。

手順

1. JVMSERVER リソースを作成し、WebSphere Liberty サーバーをサポートするように構成します。WebSphere Liberty JVMSERVER の作成方法について詳しくは、Liberty JVM サーバーの構成を参照してください。

以前に z/OS Connect for CICS 1.0 をインストールした場合は、このステップを省略できます。

2. JVM サーバー・オプションに ZCEE_INSTALL_DIR を追加します。 オプションと例について詳しくは、JVM プロファイルに使用するシンボルを参照してください。
3. セキュリティー要件のために WebSphere Liberty を構成します。デフォルトでは、クライアント認証 SSL 証明書を使用する必要があります。この HTTP 基本認証を有効にするには、 `server.xml` ファイルに次の構成オプションを追加します。

```
<!-- Allow fail-over to HTTP Basic Authentication -->
<webAppSecurity allowFailOverToBasicAuth="true"/>
```

z/OS Connect のユーザーに `zosConnectAccess` セキュリティー・ロールを付与することも必要です。WebSphere Liberty セキュリティーについて詳しくは、Liberty JVM サーバーのセキュリティーの構成 (Configuring security for a Liberty JVM server)を参照してください。z/OS Connect セキュリティーについては、z/OS Connect のセキュリティーを参照してください。

以前に z/OS Connect for CICS 1.0 をインストールした場合は、このステップを省略できます。

次のタスク

以下のタスクで説明されているように、z/OS Connect Enterprise Edition のご使用のバージョンに対応したステップを実行してください。

***CICS JSON Web** サービス用の **z/OS Connect for CICS** の構成:*

z/OS Connect for CICS を最初に構成した後に、それを CICS JSON Web サービス用に構成することができます。 JSON Web サービスは、他の CICS PIPELINE 環境にデプロイする方法と同様の方法で z/OS Connect for CICS にデプロイされます。

始める前に

このタスクを開始する前に、z/OS Connect の基本構成を完了する必要があります。z/OS Connect for CICS 1.0 については、 82 ページの『z/OS Connect for CICS 1.0の構成』を参照してください。z/OS Connect Enterprise Edition については、 89 ページの『z/OS Connect Enterprise Editionの構成』を参照してください。また、デプロイするサービスごとに JSON WSBind ファイルが必要です。このようなバインディング・ファイルは、CICS JSON アシスタント (**DFHLS2JS** および **DFHJS2LS** ユーティリティー・プログラム) を使用して生成できます。これらのユーティリティー・プログラムについて詳しくは、CICS JSON アシスタントを参照してください。

このタスクについて

サービスは z/OS Connectに、z/OS Connect 管理対象サービスとして、または CICS 管理対象サービスとしてデプロイできます。このトピックでは、これらのサービスを CICS 管理対象サービスとしてデプロイする方法について説明します。サービスを CICS 管理対象サービスとしてデプロイすることにより、サービスごとに

WEBSERVICE リソースが存在するようになります。このデプロイメント・メカニズムは、CICS の古い JSON Web サービス・テクノロジーと互換性があります。古いテクノロジーでは、サービスを z/OS Connect にデプロイするときに、SOAP Web サービスなどの他の CICS PIPELINE 環境と同様の方法でデプロイされます。

z/OS Connect Enterprise Edition V3 がある場合は、z/OS Connect Enterprise Edition V3.0 の製品資料で説明されているように、サービスを z/OS Connect 管理対象リソースとしてデプロイすることによって、より良い結果を得られる場合があります。

注: このタスクは、z/OS Connect Enterprise Edition V1.0、または CICS TS で提供される CICS サービス・プロバイダーを使用する場合に適用されます。 z/OS Connect Enterprise Edition v3.0 で提供される CICS サービス・プロバイダーを使用する場合は、z/OS Connect Enterprise Edition V3.0 製品資料内の『サービス・アーカイブ管理の自動化』を参照してください。

手順

- WSBind ファイルを適切な PIPELINE リソースに関連付ける WEBSERVICE リソースをインストールします。 WEBSERVICE には分かりやすい名前を選択してください。この名前は z/OS Connect でもサービスの名前として使用されます。必要に応じて、**PERFORM PIPELINE SCAN** コマンドを使用して WEBSERVICE リソースをインストールできます。どちらの方法を選択して使用するとしても、URIMAP に関する以下の情報を考慮するようにしてください。

  z/OS Connect 内のサービスが使用可能な URI は、次のいずれかの場所から取得されます。

  - z/OS Connect が使用するデフォルトの命名規則。
  - WSBind ファイルに保管されている URI (**PERFORM PIPELINE SCAN** コマンドを使用して関連 WEBSERVICE リソースがインストールされた場合)。
  - Liberty `server.xml` ファイル内のサービス固有の構成 (**invokeURI** 構成パラメーターが使用される場合)。
  - z/OS Connect Enterprise Edition のみ: サービスをカプセル化した API のアプリケーション・アーカイブ・ファイル (`.aar` ファイル)。

  デフォルトの命名規則を使用する場合、サービスは通常、次の URI で公開されます。

  ```
  https://<hostname>:<port>/zosConnect/services/<Service Name>?action=invoke
  ```

  この例では、`<Service Name>` はサービスの名前 (より具体的に言えば、WEBSERVICE リソースの名前) を表し、`<hostname>` および `<port>` は Liberty サーバーの構成から取得されます。

- オプション: z/OS Connect サービス用の URIMAP リソースをインストールします。 URIMAP リソースは、サービスの作業を CICS 内の特定のトランザクション ID と初期ユーザー ID に関連付けるために、CICS によって使用されます。 単一の URIMAP を使用して 1 つ以上のサービスを構成できます。z/OS Connect に対応する CICS を構成するときに、デフォルトの URIMAP リソースを定義することもできます。 z/OS Connect サービスの URI と一致する

URIMAP が存在しなければ、アプリケーションのタスクはトランザクション CJSA のもとで実行され、CICS デフォルト・ユーザー ID (通常は CICSUSER) が初期ユーザー ID として使用されます。初期ユーザー ID の役割について詳しくは、z/OS Connect サービスおよび API の許可の構成を参照してください。

URIMAP リソースを作成することを選択した場合、関連付けられたユーザー ID には指定されたトランザクションを実行する権限が必要です。

URIMAP を使用して URI を特定の WEBSERVICE リソースに関連付けることもできます。URIMAP が WEBSERVICE に密にバインドされている場合は、URIMAP によって突き合わされるすべての HTTP 要求にターゲット WEBSERVICE が使用されます。 WEBSERVICE が指定されない場合は、z/OS Connect サービスの名前に基づいて WEBSERVICE が選択されます。URIMAP の WEBSERVICE 属性がブランクのままの場合は、z/OS Connect 自らマッピングを行うので、z/OS Connect Enterprise Edition は API 内で異なるサービスにそれぞれ異なる HTTP メソッドを関連付けられるようになります。

- オプション: `server.xml` ファイルを更新します。 使用パターンによっては、Liberty サーバー構成ファイル `server.xml` の変更が必要な場合があります。通常、サービスのための特殊なケースの処理を必要としない限り、`server.xml` ファイルでサービス固有の変更を行う必要はありません。例えば、`server.xml` ファイルで z/OS Connect インターセプターの特定のセットをサービスに関連付けたり、z/OS Connect のデフォルトの命名規則と一致しない URI を使用してサービスを公開したりすることもできます。

  `server.xml` ファイルでサービスを定義するには、以下のステップを使用します。

  1. 目的とするサービス名が CICS 内の WEBSERVICE リソースの名前と一致しないことを確認します。CICS は構成情報を自動的に z/OS Connect に注入します。明示的に定義されたサービスと CICS WEBSERVICE が同じ名前の場合、結果の動作は予測不能です。
  2. `server.xml` ファイルの **`invokeURI`** 属性に、URIMAP で使用される URI と一致する適切な値を設定します。
  3. **`CICSEndpointService`** `serviceRef` エレメントにサービスをバインドします。
  4. サービスの URI を使用される厳密な WEBSERVICE に対応させる URIMAP が存在することを確認します。ステップ **2** で **`invokeURI`** 属性を設定した場合、URIMAP はその URI と一致しなければなりません。そうでなければ、URIMAP は z/OS Connect のデフォルトの URI 命名規則を前提としなければなりません。

  次の例は、`server.xml` ファイルでの明示的な z/OS Connect for CICS 1.0 サービス宣言を示しています。

```
<zosConnectService invokeURI="/json/myCustomService"
             serviceName="CICSService1"
             serviceRef="com.ibm.cics.wlp.zosconnect.CICSEndpointService"/>
```

  次の例は、z/OS Connect Enterprise Edition の場合の `server.xml` ファイルでの同等の宣言を示しています。

```
<zosconnect_zosConnectService invokeURI="/json/myCustomService"
             serviceName="CICSService1"
             serviceRef="com.ibm.cics.wlp.zosconnect.CICSEndpointService"/>
```

どちらの例も、"/json/myCustomService" URI が z/OS Connect for CICS 受信プログラム CICSService1 に関連付けられます。

タスクの結果

URL https://*hostname*:*portnumber*/zosConnect/services を Web ブラウザーに入力することによって、z/OS Connect for CICS の構成をテストできます。*hostname* は、z/OS Connect for CICS をホストする CICS 領域の IP アドレスまたは名前です。*portnumber* は、server.xml 構成ファイルの <httpEndpoint> セクションに構成されている **httpsPort** です。 Web ブラウザーには、インストール済みのサービスのリストが表示されます。

これで、同じ *hostname* と *portnumber* を使用する JSON クライアントからサービスを呼び出す準備ができました。

インストールされた各サービスの項目が zosConnect/services リストにあります。次に例を示します。

```
{
"id":"EXAMPLE",
"name":"EXAMPLE",
"url":"https://hostname:portnumber/zosConnect/services/EXAMPLE",
"protocol":"REST",
"description":"CICS Service"
}
```

この例では、関連付けられた WEBSERVICE リソースには "EXAMPLE" という名前が付けられており、このサービス定義は EXAMPLE WEBSERVICE リソースがインストールされたときに CICS によって動的に作成されています。 Web ブラウザーを使用して https://hostname:portnumber/zosConnect/services/EXAMPLE にアクセスできます。アクセスすると、サービスに関する詳細が記述された、次のような文書が返されます。

```
{
"id":"EXAMPLE",
"name":"EXAMPLE",
"protocol":"REST",
"description":"CICS Service",
"restEndpoints":[
      {
       "name":"EXAMPLE",
       "address":"hostname:portnumber/jsonTests/myExampleService"
      }
   ]
}
```

この例では、"address" はサービスが公開される URI です。この URI は、WSBind ファイル内の情報から、または **invokeURI** 属性から、あるいはデフォルトの z/OS Connect 命名規則から得られる可能性があります。

要求が Liberty JVM サーバーに到着すると、server.xml 構成ファイルの情報を使用する z/OS Connect for CICS 受信プログラムがその要求に関連付けられます。この処理を実行するために新しい CICS タスクが開始され、URIMAP の情報を使用する特定の WEBSERVICE リソースと関連付けられます。データ変換プロセスが Liberty JVM サーバー内で行われ、WSBind ファイルで指定されているターゲットの CICS プログラムが接続されます。

JSON Web サービスから ***z/OS Connect*** へのマイグレーション**:**

JSON サービスは、CICS の古い JSON Web サービス・テクノロジーから z/OS Connect に再デプロイできます。また、Mobile Extensions の古いフィーチャー・パックからも再デプロイできます。こうした再デプロイには、互換モードで z/OS Connect を構成する必要があります。このモードは z/OS Connect for CICS 1.0 でも z/OS Connect Enterprise Edition でも使用できます。

このタスクについて

このタスクで *z/OS Connect* とは、z/OS Connect for CICS 1.0 と z/OS Connect Enterprise Edition の両方を指します。

Java Pipelines for JSON (CICS Transaction Server Feature Pack for Mobile Extensions V1.0 で使用) に含まれる JSON Web サービスを z/OS Connect に再デプロイする方法が、プロセスとして簡単です。 Java Pipelines for JSONと z/OS Connect に使用される WSBind ファイルは、同じツール (DFHLS2JS および DFHJS2LS) を使用して生成され、相互に完全な互換性があります。CICS でこの機能を探索するなら、IBM Redbooks: Implementing IBM CICS JSON Web Services for Mobile Applications に JSON Web サービスの例があります。

z/OS Connect 環境では、クライアント・アプリケーションと CICS の間で SSL を使用することをお勧めします。既存の Java Pipelines for JSON環境で SSL を使用していない場合は、z/OS Connect への移行には追加の手順を必要とします。

手順

1. z/OS Connect for CICS の構成の手順を使用して、必要な z/OS Connect インフラストラクチャーを作成します。この構成の一環として、Liberty JVM サーバーが接続の着信を listen する SSL TCP/IP ポート番号を選択します。次の 2 つのオプションから検討します。
   a. Java Pipelines for JSONの TCPIPSERVICE によって使用されるポート番号とは異なるポート番号を選択します。このオプションは、両方の環境をそれぞれ異なる TCP/IP ポートに同時にインストールできるという利点があります。これは、新しい JSON サービスを対象とするようにクライアント・プログラムを更新する必要があることを意味します。以前の環境で SSL を使用していなかった場合は、z/OS Connect への移行で URI に変更を加える必要があるため、このオプションの方が適しています。
   b. Java Pipelines for JSONの TCPIPSERVICE で使用しているのと同じポート番号を選択します。クライアント・プログラムによって使用される URI でポート番号を変更する必要はありませんが、2 つの環境を同時にインストールすることはできません。 URI は、その他の理由により変更を加える必要がある場合があります。例えば、SSL を有効にするときの HTTP から HTTPS への切り替えです。
2. WSBind ファイルを z/OS Connect にデプロイします。既存の WSBind ファイルは、z/OS Connect と完全に互換性があります。 CICS JSON Web サービス用の z/OS Connect の構成に記載されている手順に従って、新しい JSON Web サービスを z/OS Connect にデプロイします。 CICS では、同じ名前の 2 つの WEBSERVICE リソースを両方インストールすることはできません。したがって、次のいずれかを行います。

a. 新しい WEBSERVICE を以前のものと同じ名前でインストールするために、元の WEBSERVICE を破棄します。
b. 競合を避けるために、新しい WEBSERVICE の名前を変更します。

3. PIPELINE ハンドラー・プログラムを使用して Java Pipelines for JSONの処理をカスタマイズした場合、そのカスタマイズが引き続き必要かどうか検討します。必要な場合は、z/OS Connect Interceptor プログラムを相当する関数で作成して、それらをグローバル・インターセプターとしてデプロイします。 z/OS Connect Interceptor について詳しくは、 z/OS Connect インターセプターの定義を参照してください。

タスクの結果

これで、新しい z/OS Connect JSON Web サービスをテストする準備ができました。Java Pipelines for JSONで使用したポート番号と同じポート番号を使用してサービスをデプロイした場合、クライアントに変更を加える必要はありません (ただし、SSL を有効にするなどのセキュリティー構成を変更した場合を除きます)。ポート番号またはサービスの URI を変更した場合は、クライアントに変更を加える必要があります。

z/OS Connect Enterprise Edition からの ***API*** の使用*:*

z/OS Connect Enterprise Edition の主な機能の一つは、1 つ以上の JSON サービスから JSON API を構成できることです。この機能は z/OS Connect for CICS 1.0 には存在しません。最適な API 開発環境を整えるためには、古い JSON Web サービス・テクノロジーを CICS で互換モードで使用するのではなく、API とサービスを z/OS Connect Enterprise Edition v3.0 にデプロイしてください。詳しくは、z/OS Connect Enterprise Edition の資料を参照してください。

API Toolkit の詳細については、z/OS Connect Enterprise Edition V3.0 の製品資料を参照してください。

CICS 用 z/OS Connect Enterprise Edition への API のデプロイは、z/OS Connect Enterprise Edition のスタンドアロン・インストール済み環境への API のデプロイに比べて、多少の違いがあります。CICS への API のデプロイには、以下の考慮事項が適用されます。

- API 用の既存の WSBind ファイルを取り入れることはできません。API Editor は、その入力の一部として、1 つ以上のサービス・アーカイブ・リソース (SAR) ファイルを必要とします。

  CICS TS で提供される CICS サービス・プロバイダーを使用する場合、z/OS Connect Enterprise Edition と一緒に配布される BAQLS2JS アシスタントまたは BAQJS2LS アシスタントを使用して WSBind ファイルが作成されます。API を作成する場合は、z/OS Connect Enterprise Edition に付属しているアシスタントを使用して、必要な WSBind および SAR ファイルを生成することから始めます。

  CICS で使用される WEBSERVICE リソースの名前に合わせて、BAQLS2JS または BAQJS2LS の **SERVICE-NAME** パラメーターを設定する必要があります。 CICS は、z/OS Connect でのサービスの名前と CICS での WEBSERVICE の名前が 1 対 1 で一致することを必要とします。

z/OS Connect Enterprise Edition で提供される CICS サービス・プロバイダーを使用する場合、z/OS Connect Enterprise Edition と一緒に配布される z/OS Connect Enterprise Edition V3 API Toolkit またはビルド・ツールキットのいずれかのサービス・エディターを使用して、SAR ファイルが作成されます。API を作成するには、その前にこれらのツールを使用して必要な SAR ファイルを生成する必要があります。

- サービスの名前と一致する WEBSERVICE リソース定義が CICS に必要です。

  CICS TS で提供される CICS サービス・プロバイダーを使用すると、WEBSERVICE によってサービスの WSBind ファイルがカプセル化されます。

  z/OS Connect EE V3 で提供される CICS サービス・プロバイダーを使用する場合、WEBSERVICE リソース定義は不要です。

-

  URIMAP リソース定義も CICS で提供される場合があります。URIMAP が使用されると、CICS は自動的に API のための作業をトランザクション ID に関連付け、適切な初期ユーザー ID を設定します。

- API Editor は、アプリケーション・アーカイブ (AAR) ファイルを出力として生成します。これは、z/OS Connect EE が提供するデプロイメント・メカニズムを使用して、z/OS Connect EE サーバーにデプロイする必要があります。デフォルトでは、API はサーバーの構成ディレクトリーの `/resources/zosconnect/apis` ディレクトリーにデプロイされます。このディレクトリーがまだ存在していない場合、これを作成する必要があります。あるいは、API 用に代わりのディレクトリーを作成して、それを `server.xml` の `zosconnect_APIs` エレメントで指定することもできます。

上記の考慮事項に従えば、API とそのコンポーネント・サービスが CICS にデプロイされます。適切な URIMAP および WEBSERVICE リソースが CICS に構成され、Liberty 構成作業域に AAR ファイルがデプロイされます。Liberty のメイン構成ファイルである `server.xml` には、デプロイされた各コンポーネント・サービスの項目が含まれます。

z/OS Connect* 管理対象サービスを使用するための *z/OS Connect Enterprise Edition V3.0* の構成*:

z/OS Connect Enterprise Edition は個別注文の製品であり、CICS TS の一部として提供される製品ではありません。まず、z/OS Connect Enterprise Edition ランタイム・コンポーネントをインストールし、CICS がそれを見つけることができるように zFS ファイルを構成する必要があります。次に、z/OS Connect 用の JVM サーバーを構成し、パイプライン構成およびリソースをセットアップします。その後、JSON サービスおよび API をデプロイできるようになります。この初期構成は 1 回限りのアクティビティーであり、CICS TS とともに配布される Liberty サーバーに z/OS Connect Enterprise Edition を組み込んで実行できるようにするための構成です。

始める前に

z/OS Connect Enterprise Edition ランタイムをインストールします。 z/OS Connect Enterprise Edition V3.0 の製品資料の説明に従ってください。z/OS

Connect Enterprise Edition を CICS 内に組み込んで実行する準備をする場合、別個の Liberty サーバー・インスタンスを作成する必要はありません。 zFS には少なくともファイル・システム・コンポーネントをインストールする必要があります。 z/OS Connect Enterprise Edition で提供される API Toolkit をインストールする作業は、このタスクの一部ではありません。

z/OS Connect for CICS 1.0 が既にインストールされていますか? そうであれば、このタスクのいくつかのステップは必要ありません。スキップできるステップはリストの中で示しています。

手順

1. 89 ページの『z/OS Connect Enterprise Editionの構成』 のすべてのステップが完了していることを確認します。
2. server.xml ファイル内の <featureManager> リストを更新して、zosconnect:cicsService-1.0 機能を組み込みます。 その例を次に示します。

   ```
   <featureManager>
        <feature>cicsts:core-1.0</feature>
        <feature>zosconnect:cicsService-1.0</feature>
   </featureManager>
   ```

   注: CICS TS によって提供される CICS サービス・プロバイダーと z/OS Connect Enterprise Edition v3.0 によって提供される CICS サービス・プロバイダーを同時に使用することはできません。 z/OS Connect for CICS 1.0 から z/OS Connect Enterprise Edition V3.0 にアップグレードする場合は、フィーチャー・コードを変更する必要があります。
3. CICS サービス・プロバイダーのエレメントを定義します。 z/OS Connect for CICS 1.0 をインストール済みの場合、すでに適切に構成されていると思われるため、この手順をスキップしてもかまいません。

   次のエレメントを server.xml で構成します。

   a. ローカル CICS 接続またはリモート IPIC 接続のいずれかを定義します。 以下の定義例は、ローカル CICS 接続の場合です。

      ```
      <zosconnect_cicsLocalConnection id="eciTest"/>
      ```

      次の定義例は、リモート IPIC 接続の場合です。

      ```
      <zosconnect_cicsIPICConnection
          id="eciTest"
          host="cicshost.company.com"
          port="1111"/>
      ```

   b. SAR ファイルを使用してサービスのデプロイメントを可能にします。 その例を次に示します。

      ```
      <zosconnect_services
                  pollingRate="5s"
                  updateTrigger="polled">
      ```

   c. セキュリティーを無効にするには、zosconnect_zosConnectManager を構成します。

   z/OS Connect Enterprise Edition で提供される CICS サービス・プロバイダーの構成について詳しくは、CICS サービス・プロバイダーの使用を参照してください。

4. JVMSERVER をインストールします。生成された messages.log ファイルで エラー・メッセージあるいは警告メッセージがないか確認します。 このログには、WebSphere Liberty Server によって生成されるメッセージが格納されます。その中には z/OS Connect Enterprise Edition によって発行される次のようなメッセージも含まれます。

```
SRVE0169I: Web モジュールをロード中: z/OS Connect。
SRVE0250I: Web モジュール z/OS Connect は、default_host にバインドされています。
```

タスクの結果

z/OS Connect Enterprise Edition インスタンスが構成されました。 URL `https://hostname:portnumber/zosConnect/services` を Web ブラウザーに入力して z/OS Connect の基本構成をテストできます。ここで、*hostname* は z/OS Connect をホストしている CICS 領域が実行されているシステムのIP アドレスまたはホスト名で、*portnumber* は server.xml ファイルの <httpEndpoint> 要素で指定された **httpsPort** です。Web ブラウザーにはインストール済みのサービスのリストが表示されますが、インストールされているサービスがまだないため、リストは空です。

予想されるサービス・リストではなく HTTP 403 AuthorizationFailed 応答を受け取った場合は、 89 ページの『z/OS Connect Enterprise Editionの構成』のセキュリティー構成ステップを確認してください。認証済みユーザーが z/OS Connect の使用を許可されていない可能性があります。

次のタスク

これで、JSON サービスまたは API を z/OS Connect Enterprise Edition にデプロイする準備ができました。サービスのデプロイについて詳しくは、z/OS Connect Enterprise Edition V3.0 製品資料内の『z/OS 資産の REST API としての公開』を参照してください。API のデプロイについて詳しくは、 95 ページの『z/OS Connect Enterprise Edition からの API の使用』を参照してください。

***CICS* 管理対象サービスを使用するための *z/OS Connect Enterprise Edition* の構成:**

z/OS Connect Enterprise Edition は個別注文の製品であり、CICS TS の一部として提供される製品ではありません。まず、z/OS Connect Enterprise Edition ランタイム・コンポーネントをインストールし、CICS がそれを見つけることができるように zFS ファイルを構成する必要があります。次に、z/OS Connect 用の JVM サーバーを構成し、パイプライン構成およびリソースをセットアップします。その後、JSON サービスおよび API をデプロイできるようになります。この初期構成は 1 回限りのアクティビティーであり、CICS TS とともに配布される Liberty サーバーに z/OS Connect Enterprise Edition を組み込んで実行できるようにするための構成です。

始める前に

z/OS Connect Enterprise Edition ランタイムをインストールします。 z/OS Connect Enterprise Edition V2.0 の製品資料の説明に従ってください。z/OS Connect Enterprise Edition を CICS 内に組み込んで実行する準備をする場合、別個の Liberty サーバー・インスタンスを作成する必要はありません。 zFS には少な

くともファイル・システム・コンポーネントをインストールする必要があります。 z/OS Connect Enterprise Edition で提供される API Toolkit をインストールする作業は、このタスクの一部ではありません。

z/OS Connect for CICS 1.0 が既にインストールされていますか? そうであれば、このタスクのいくつかのステップは必要ありません。スキップできるステップはリストの中で示しています。

手順

1. 89 ページの『z/OS Connect Enterprise Editionの構成』 のすべてのステップが完了していることを確認します。
2. server.xml ファイル内の <featureManager> リストを更新して、cicsts:zosConnect-2.0 機能を組み込みます。 その例を次に示します。

   ```
   <featureManager>
       <feature>cicsts:core-1.0</feature>
       <feature>transportSecurity-1.0</feature>
       <feature>cicsts:zosConnect-2.0</feature>
   </featureManager>
   ```

   注: z/OS Connect for CICS 1.0 の機能と z/OS Connect Enterprise Edition の機能を同時に使用することはできません。 z/OS Connect for CICS 1.0 から z/OS Connect Enterprise Edition にアップグレードする場合は、この機能を変更する必要があります。
3. z/OS Connect Service Controller を定義します。 CICS TS で提供される CICS サービス・プロバイダーを使用する場合は、以下のステートメントを server.xml ファイルに追加します。

   ```
   <com.ibm.cics.wlp.zosconnect.CICSEndpoint
       id="com.ibm.cics.wlp.zosconnect.CICSEndpointService"/>
   ```

   z/OS Connect for CICS 1.0 をインストール済みの場合、すでに適切に構成されていると思われるため、この手順をスキップしてもかまいません。

   z/OS Connect Enterprise Edition V2 で提供される CICS サービス・プロバイダーを使用する場合は、server.xml で以下のエレメントを構成します。

   a. ローカル CICS 接続またはリモート IPIC 接続のいずれかを定義します。 以下の定義例は、ローカル CICS 接続の場合です。

      ```
      <zosconnect_cicsLocalConnection id="eciTest"/>
      ```

      次の定義例は、リモート IPIC 接続の場合です。

      ```
      <zosconnect_cicsIPICConnection
          id="eciTest"
          host="cicshost.company.com"
          port="1111"/>
      ```

   b. SAR ファイルを使用してサービスのデプロイメントを可能にします。 その例を次に示します。

      ```
      <zosconnect_services
          pollingRate="5s"
          updateTrigger="polled">
      ```

z/OS Connect Enterprise Edition の CICS サービス・プロバイダーの構成について詳しくは、z/OS Connect Enterprise Edition V3.0 製品資料内の『CICS サービス・プロバイダーの使用』を参照してください。

4. JVMSERVER をインストールします。生成された messages.log ファイルで エラー・メッセージあるいは警告メッセージがないか確認します。 このログには、WebSphere Liberty Server によって生成されるメッセージが格納されます。その中には z/OS Connect Enterprise Edition によって発行される次のようなメッセージも含まれます。

   ```
   SRVE0169I: Web モジュールをロード中: z/OS Connect。
   SRVE0250I: Web モジュール z/OS Connect は、default_host にバインドされています。
   ```

5. XML パイプライン構成ファイルを作成します。サンプルのパイプライン構成ファイル jsonzosconnectprovider.xml がディレクトリー/usr/lpp/cicsts/cicsts55/samples/pipelines/ (ここで /usr/lpp/cicsts/cicsts55 は z/OS UNIX の CICS ファイルのデフォルト・インストール・ディレクトリー) にあります。Liberty JVM サーバー内で Java を使用して JSON を解析するか (デフォルト)、Java 以外の JSON パーサーを使用するかを判断します。
   - Liberty JVM サーバー内で Java を使用して JSON を解析するには、サンプルのパイプライン構成ファイルを使用できます。その際、<jvmserver> 要素の DFHWLP をステップ 2 の JVMSERVER の名前に置き換えてください。
   - Java 以外のパーサーを使用して JSON を解析するには、サンプルの構成ファイルを変更して、以下の例のように java_parser="no" 属性を <provider_pipeline_json> 要素に追加します。

```
<?xml version="1.0" encoding="EBCDIC-CP-US"?>
<provider_pipeline_json java_parser="no" xmlns="http://www.ibm.com/software/htp/cics/pipeline">
    <jvmserver>DFHWLP</jvmserver>
</provider_pipeline_json>
```

   DFHWLP をこのプロシージャーの開始時に作成した JVMSERVER の名前と置き換えます。

   z/OS Connect for CICS 1.0 をインストール済みの場合、すでに適切に構成されていると思われるため、この手順をスキップしてもかまいません。

6. パイプライン構成ファイルを zFS 内の適切なディレクトリーにコピーし、CICS 領域がファイルを読み取れるようなファイル権限になっていることを確認します。 詳しくは、『パイプライン構成ファイル』を参照してください。

   z/OS Connect for CICS 1.0 をインストール済みの場合、すでに適切に構成されていると思われるため、この手順をスキップしてもかまいません。

7. PIPELINE リソースを作成します。 PIPELINE リソースは、パイプライン構成ファイルの場所を定義します。 WEBSERVICE をこの PIPELINE にインストールするために SCAN メカニズムを使用しないでください。

   z/OS Connect for CICS 1.0 をインストール済みの場合、すでに適切に構成されていると思われるため、この手順をスキップしてもかまいません。

8. オプション: z/OS Connect 用のデフォルトの URIMAP リソースを作成します。 URIMAP リソースは、トランザクションおよびデフォルト・ユーザー ID を z/OS Connect の作業に関連付けるために使用されます。1 つ以上の URIMAP リソースを使用して、z/OS Connect のデフォルト・ポリシーを構成

できます。 URIMAP の構成例、および構成オプションの詳細については、z/OS Connect サービスおよび API のアクセス権の構成 を参照してください。

注:

z/OS Connect は、HTTP 要求ごとに別の認証を行います。その結果、CICS で実行されるアプリケーション・タスクは通常、URIMAP からの初期ユーザー ID よりも限定的なユーザー ID に関連付けられます。

この初期ユーザー ID が有効なのは、z/OS Connect 内でユーザー固有の認証が行われるまでに限られます。デフォルトの CICS ユーザー ID (通常は「CICSUSER」) を使用してそれにターゲット・トランザクションを使用する権限を与えるのではなく、ZOSCUSER などの特定のユーザー ID を URIMAP で使用してください。

タスクの結果

z/OS Connect Enterprise Edition インスタンスが構成されました。 URL `https://hostname:portnumber/zosConnect/services` を Web ブラウザーに入力して z/OS Connect の基本構成をテストできます。ここで、*hostname* は z/OS Connect をホストしている CICS 領域が実行されているシステムのIP アドレスまたはホスト名で、*portnumber* は `server.xml` ファイルの `<httpEndpoint>` 要素で指定された **`httpsPort`** です。Web ブラウザーにはインストール済みのサービスのリストが表示されますが、インストールされているサービスがまだないため、リストは空です。

予想されるサービス・リストではなく `HTTP 403 AuthorizationFailed` 応答を受け取った場合は、 89 ページの『z/OS Connect Enterprise Editionの構成』のセキュリティー構成ステップを確認してください。認証済みユーザーが z/OS Connect の使用を許可されていない可能性があります。

次のタスク

これで、JSON サービスまたは API を z/OS Connect Enterprise Edition にデプロイする準備ができました。サービスのデプロイについて詳しくは、CICS JSON Web サービス用の z/OS Connect の構成を参照してください。API のデプロイについて詳しくは、 95 ページの『z/OS Connect Enterprise Edition からの API の使用』を参照してください。

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。

パイプライン構成ファイルは、z/OS UNIX システム・サービスのファイル・システムに保管されます。このファイルの名前は、PIPELINE リソース定義の 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。 パイプライン構成ファイルの XML スキーマは、ディレクトリー `/usr/lpp/cicsts/cicsts55/schemas/pipeline/` (`/usr/lpp/cicsts/cicsts55` は z/OS UNIX 上の CICS

ファイルのデフォルト・インストール・ディレクトリー) にあります。構成ファイルを操作するときには、文字セット・エンコード方式が UTF-8 であることを確認してください。 EBCDIC でエンコードされた既存の構成ファイルをインポートすると、自動的に UTF-8 に変換されます。

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。 Web サービス・セキュリティーのサポートや Web サービス・トランザクションなど、さまざまなカテゴリーのアプリケーションに適用される実行環境の局面を提供するために、パイプラインが構成されます。一般に、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用することが可能です。

注: CICS Explorer を使って新しい PIPELINE 構成ファイルをバンドルの一部として作成するとき、バンドルのルートに同じ名前の構成ファイルが存在してはなりません。

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> エレメントには、次のような隣接したサブエレメントがあります。

- <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。 このエレメントは、<provider_pipeline> エレメント内で使用する際は必須で、<requester_pipeline> エレメント内ではオプションです。
- 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。
- オプションの <apphandler> エレメント (<provider_pipeline> の場合のみ)。これは、チャネル接続アプリケーション・ハンドラーを指定するのに使用されます。

- オプションの <apphandler_class> エレメント (<provider_pipeline> の場合のみ)。これは、Axis2 アプリケーション・ハンドラーを指定するのに使用されます。
- 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含みます。

一部のエレメントは、そのエレメントに関連付けられる属性を持つことがあります。 有効な XML 文書を作成するには、各属性値を引用符で囲む必要があります。

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、z/OS UNIX 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、独自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。これらのファイルは、ライブラリー /usr/lpp/cicts/cicsts55/samples/pipelines にあります。

basicsoap11provider.xml
: Java をサポートしないパイプラインで SOAP 1.1 プロトコルを使用するサービス・プロバイダー・パイプライン定義。このパイプラインは、<cics_soap_1.1_handler> メッセージ・ハンドラーを使用し、CICS Web サービス・アシスタントを使用して CICS アプリケーションが配置されているときに使用されます。

basicsoap11requester.xml
: Java をサポートしないパイプラインで SOAP 1.1 プロトコルを使用するサービス・リクエスター・パイプライン定義。このパイプラインは、<cics_soap_1.1_handler> メッセージ・ハンドラーを使用し、CICS Web サービス・アシスタントを使用して CICS アプリケーションが配置されているときに使用されます。

basicsoap11javaprovider.xml
: Java をサポートするパイプラインで SOAP 1.1 プロトコルを使用するサービス・プロバイダー・パイプライン定義。このパイプラインは、<cics_soap_1.1_handler_java> メッセージ・ハンドラーを使用し、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が配置されているときに使用されます。 この構成にはエレメント <jvmserver> が含まれます。この構成を使用するには、その前にこのメッセージ・ハンドラーを編集し、適切な JVM サーバーを指定する必要があります。

basicsoap11javarequester.xml
: Java をサポートするパイプラインで SOAP 1.1 プロトコルを使用するサービス・リクエスター・パイプライン定義。このパイプラインは、<cics_soap_1.1_handler_java> メッセージ・ハンドラーを使用し、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が配置されているときに使用されます。 この構成にはエレメント <jvmserver> が含まれます。この構成を使用するには、その前にこのメッセージ・ハンドラーを編集し、適切な JVM サーバーを指定する必要があります。

jsonjavaprovider.xml
: Java をサポートするパイプライン用の JSON メッセージ・フォーマットを

使用するサービス・プロバイダー・パイプライン定義。このパイプラインは、<cics_json_handler_java> メッセージ・ハンドラーを使用し、CICS JSON アシスタントを使用して CICS アプリケーションが配置されているときに使用されます。この構成にはエレメント <jvmserver> が含まれます。この構成を使用するには、その前にこのメッセージ・ハンドラーを編集し、適切な JVM サーバーを指定する必要があります。

jsonzosconnectprovider.xml

z/OS Connect for CICS 用に構成された PIPELINE にデプロイされた JSON Web サービスのパイプライン定義。このパイプラインは、<provider_pipeline_json> メッセージ・ハンドラーを使用します。この構成にはエレメント <jvmserver> が含まれます。この構成を使用するには、その前にこのメッセージ・ハンドラーを編集し、適切な JVM サーバーを指定する必要があります。

kerberosprovider.xml

Kerberos サポートについての構成情報を basicsoap11provider.xml に追加するサービス・プロバイダー・パイプライン定義。

samlprovider.xml

SAML サポートについての構成情報を basicsoap11provider.xml に追加するサービス・プロバイダー・パイプライン定義。

samlrequester.xml

SAML サポートについての構成情報を basicsoap11requester.xml に追加するサービス・リクエスター・パイプライン定義。

propagatesamlprovider.xml

SAML 情報を CICS トランザクションを通して伝搬しながら、SAML サポートの構成情報を basicsoap11provider.xml に追加するサービス・プロバイダー・パイプライン定義。

propagatesamlrequester.xml

SAML 情報を CICS トランザクションを通して伝搬しながら、SAML サポートの構成情報を basicsoap11requester.xml に追加するサービス・リクエスター・パイプライン定義。

wsatprovider.xml

Web サービス・トランザクションについての構成情報を basicsoap11provider.xml に追加するパイプライン定義。

wsatrequester.xml

Web サービス・トランザクションについての構成情報を basicsoap11requester.xml に追加するパイプライン定義。

プロバイダー・パイプライン構成ファイルの例 (JSON アプリケーション・ハンドラー)

以下に、<cics_json_handler_java> エレメントを使用するサービス・プロバイダー・パイプライン用の、単純な構成ファイルの例を示します。

```
<p><?xml version="1.0" encoding="EBCDIC-CP-US"?>
<provider_pipeline xmlns="http://www.ibm.com/software/htp/cics/pipeline">
  <service>
```

```
    <terminal_handler>
      <cics_json_handler_java>
        <jvmserver>DFHAXIS</jvmserver>
        <repository>/usr/lpp/cicsts/cicsts52/lib/pipeline/repository</repository>
      </cics_json_handler_java>
    </terminal_handler>
  </service>
  <apphandler_class>com.ibm.cicsts.axis2.CICSAxis2ApplicationHandler</apphandler_class>
</provider_pipeline>
```

このパイプラインに含まれるメッセージ・ハンドラーは 1 つだけです。 このハンドラーはプログラム DFHJSON にリンクします。

- <provider_pipeline> エレメントは、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ルート・エレメントです。
- <service> エレメントは、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。 この例では、メッセージ・ハンドラーは 1 つしかありません。
- <terminal_handler> エレメントは、パイプラインの端末メッセージ・ハンドラーの定義を含みます。
- <cics_json_handler_java> エレメントは、パイプラインが Java ベースのパイプラインで、パイプラインのサービス・ハンドラーが JSON メッセージをサポートするメッセージ・ハンドラーであることを示します。
- <apphandler> エレメントは、パイプラインの端末ハンドラーがデフォルトでリンクするアプリケーション・ハンドラーの名前を指定します。 このケースでは、プログラムは DFHJSON で、CICS JSON アシスタントを使用して配置されたアプリケーション用の、CICS 提供のプログラムです。

プロバイダー・パイプライン構成ファイルの例 (チャネル接続アプリケーション・ハンドラー)

以下に、<cics_soap_1.1_handler> エレメントを使用するサービス・プロバイダー・パイプライン用の、単純な構成ファイルの例を示します。

```
<?xml version="1.0" encoding="EBCDIC-CP-US"?>
<provider_pipeline
   xmlns="http://www.ibm.com/software/htp/cics/pipeline"
    <service>
    <terminal_handler>
      <cics_soap_1.1_handler/>
    </terminal_handler>
  </service>
  <apphandler>DFHPITP</apphandler>
</provider_pipeline>
```

このパイプラインに含まれるメッセージ・ハンドラーは 1 つだけです。 このハンドラーはプログラム DFHPITP にリンクします。

- <provider_pipeline> エレメントは、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ルート・エレメントです。
- <service> エレメントは、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。 この例では、メッセージ・ハンドラーは 1 つしかありません。
- <terminal_handler> エレメントは、パイプラインの端末メッセージ・ハンドラーの定義を含みます。

- <cics_soap_1.1_handler> エレメントは、パイプラインが Java ベースのパイプラインでなく、パイプラインの端末ハンドラーが SOAP 1.1 メッセージをサポートするメッセージ・ハンドラーであることを示します。
- <apphandler> エレメントは、パイプラインの端末ハンドラーがデフォルトでリンクするアプリケーション・ハンドラーの名前を指定します。 このケースでは、プログラムは DFHPITP で、CICS Web サービス・アシスタントを使用して配置されたアプリケーション用の、CICS 提供のプログラムです。

プロバイダー・パイプライン構成ファイルの例 (Axis2 アプリケーション・ハンドラー)

以下に、<cics_soap_1.1_handler_java> エレメントを使用するサービス・プロバイダー・パイプライン用の、単純な構成ファイルの例を示します。

```
<?xml version="1.0" encoding="EBCDIC-CP-US"?>
<provider_pipeline
   xmlns="http://www.ibm.com/software/htp/cics/pipeline"
    <service>
    <terminal_handler>
      <cics_soap_1.1_handler_java>
      <jvmserver>DFHAXIS</jvmserver>
      </cics_soap_1.1_handler_java>
    </terminal_handler>
  </service>
  <apphandler_class>com.ibm.cicsts.axis2.CICSAxis2ApplicationHandler</apphandler_class>
</provider_pipeline>
```

このパイプラインに含まれるメッセージ・ハンドラーは 1 つだけです。 このハンドラーはプログラム DFHPITP にリンクします。

- <provider_pipeline> エレメントは、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ルート・エレメントです。
- <service> エレメントは、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。 この例では、メッセージ・ハンドラーは 1 つしかありません。
- <terminal_handler> エレメントは、パイプラインの端末メッセージ・ハンドラーの定義を含みます。
- <cics_soap_1.1_handler_java> エレメントは、パイプラインが Java ベースのパイプラインで、パイプラインのサービス・ハンドラーが SOAP 1.1 メッセージをサポートするメッセージ・ハンドラーであることを示します。
- <apphandler_class> エレメントは、提供されている Axis2 アプリケーション・ハンドラーを指定します。

リクエスター・パイプライン構成ファイルの例

Axis2 MTOM/XOP サポートを組み込んだ <cics_soap_1.2_handler_java> エレメントを使用するサービス・リクエスター・パイプラインの簡単な構成ファイルの例を以下に示します。

```
<?xml version="1.0" encoding="EBCDIC-CP-US"?>
<requester_pipeline
  xmlns="http://www.ibm.com/software/htp/cics/pipeline">
   <service>
     <service_handler_list>
       <cics_soap_1.2_handler_java>
         <jvmserver>JVMSERV1</jvmserver>
         <mtom>
```

```
      </cics_soap_1.2_handler_java>
    </service_handler_list>
  </service>
</requester_pipeline>
```

このパイプラインに含まれるメッセージ・ハンドラーは 1 つだけです。

- <requester_pipeline> エレメントは、サービス・リクエスター・パイプライン用のパイプライン構成ファイルのルート・エレメントです。
- <service> エレメントは、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。 この例では、メッセージ・ハンドラーは 1 つしかありません。
- <service_handler_list> は、すべての要求に対して呼び出されるメッセージ・ハンドラーのリストを指定します。
- <cics_soap_1.2_handler_java> エレメントは、パイプラインが Java をサポートしており、パイプラインのサービス・ハンドラーが SOAP 1.2 メッセージをサポートするメッセージ・ハンドラーであることを示します。
- <jvmserver> エレメントは、使用される JVM サーバーを指定します。
- <mtom/> エレメントでは、アウトバウンド XOP 文書を MTOM メッセージにパックして送信する、という動作を指定しています。 Java ベースのパイプラインでは、デフォルトでインバウンド MTOM メッセージが受け入れられ、アンパックされます。

z/OS Connect for CICS JSON Web サービスのプロバイダー・パイプライン構成ファイルの例

以下に、<provider_pipeline_json> エレメントを使用するサービス・プロバイダー・パイプラインの構成ファイルの簡単な例を示します。 java_parser="NO" 属性が提供されているため、非 Java JSON パーサーが使用されます。

```
<?xml version="1.0" encoding="EBCDIC-CP-US"?>
<provider_pipeline_json java_parser="NO"
  xmlns="http://www.ibm.com/software/htp/cics/pipeline">
   <jvmserver>DFHWLP</jvmserver>
</provider_pipeline_json>
```

<provider_pipeline_json> エレメントは、ハンドラー・プログラムを定義できないという点で <provider_pipeline> エレメントと異なります。

- <provider_pipeline_json> エレメントは、z/OS Connect for CICS JSON Web サービス・プロバイダー・パイプラインのパイプライン構成ファイルのルート・エレメントです。
- java_parser="NO" 属性は、非 Java JSON パーサーが使用されることを指定します。
- <jvmserver> エレメントは、使用される JVM サーバーを指定します。

注: z/OS Connect for CICS 以外のものを使用して <provider_pipeline_json> パイプラインを開始しようとしても、エラーが発生します。

非 Java JSON 構文解析のためのプロバイダー・パイプライン構成ファイルの例

```
<?xml version="1.0" encoding="EBCDIC-CP-US"?>
<provider_pipeline
    xmlns="http://www.ibm.com/software/htp/cics/pipeline">
  <service>
```

```
    <terminal_handler>
      <handler>
          <program>DFHPIJT</program><handler_parameter_list/>
      </handler>
    </terminal_handler>
  </service>
</provider_pipeline>
```

このパイプラインに含まれるメッセージ・ハンドラーは 1 つだけです。

- <provider_pipeline> エレメントは、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ルート・エレメントです。
- <service> エレメントは、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。 この例では、メッセージ・ハンドラーは 1 つだけです。
- <terminal_handler> エレメントは、パイプラインの端末メッセージ・ハンドラーの定義を含みます。
- <handler> エレメントは、ハンドラーの詳細を指定します。
- <program> エレメントは、呼び出されるプログラムを指定します。 DFHPIJT は、非 Java JSON 処理のための CICS 提供のハンドラーです。

トランスポート関連ハンドラー

各パイプラインの構成ファイルに、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てい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 WebSphere 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。 例えば、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 WebSphere 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、ビジネス・パートナーの場所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。

サービス・プロバイダーでは、インバウンド・メッセージは、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) と関連付けられます。 特定のリソースがインバウンド要求について使用される場合にのみ、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できます。

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呼び出されるメッセージ・ハンドラーを指定します。

トランスポート・セクション
: 使用中のトランスポート・リソースに応じて呼び出される、または呼び出されない、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・ハンドラーを呼び出すように決定している場合でも、これより前のメッセージ・ハンドラーによってこの決定を無効にできます。

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 接続について 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、ハンドラー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><program>HANDLER1</program><handler_parameter_list/></handler>
    <handler><program>HANDLER2</program><handler_parameter_list/></handler>
  </default_transport_handler_list>

  <!-- HANDLER3 overrides defaults for MQ transport -->
  <default_mq_transport_handler_list>
    <handler><program>HANDLER3</program><handler_parameter_list/></handler>
  </default_mq_transport_handler_list>

  <!-- HANDLER4 overrides defaults for http transport with TCPIPSERVICE(WS00) -->
  <named_transport_entry type="http">
    <name>WS00</name>
    <transport_handler_list>
      <handler><program>HANDLER4</program><handler_parameter_list/></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、z/OS UNIX 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 111 ページの図 20 に示します。

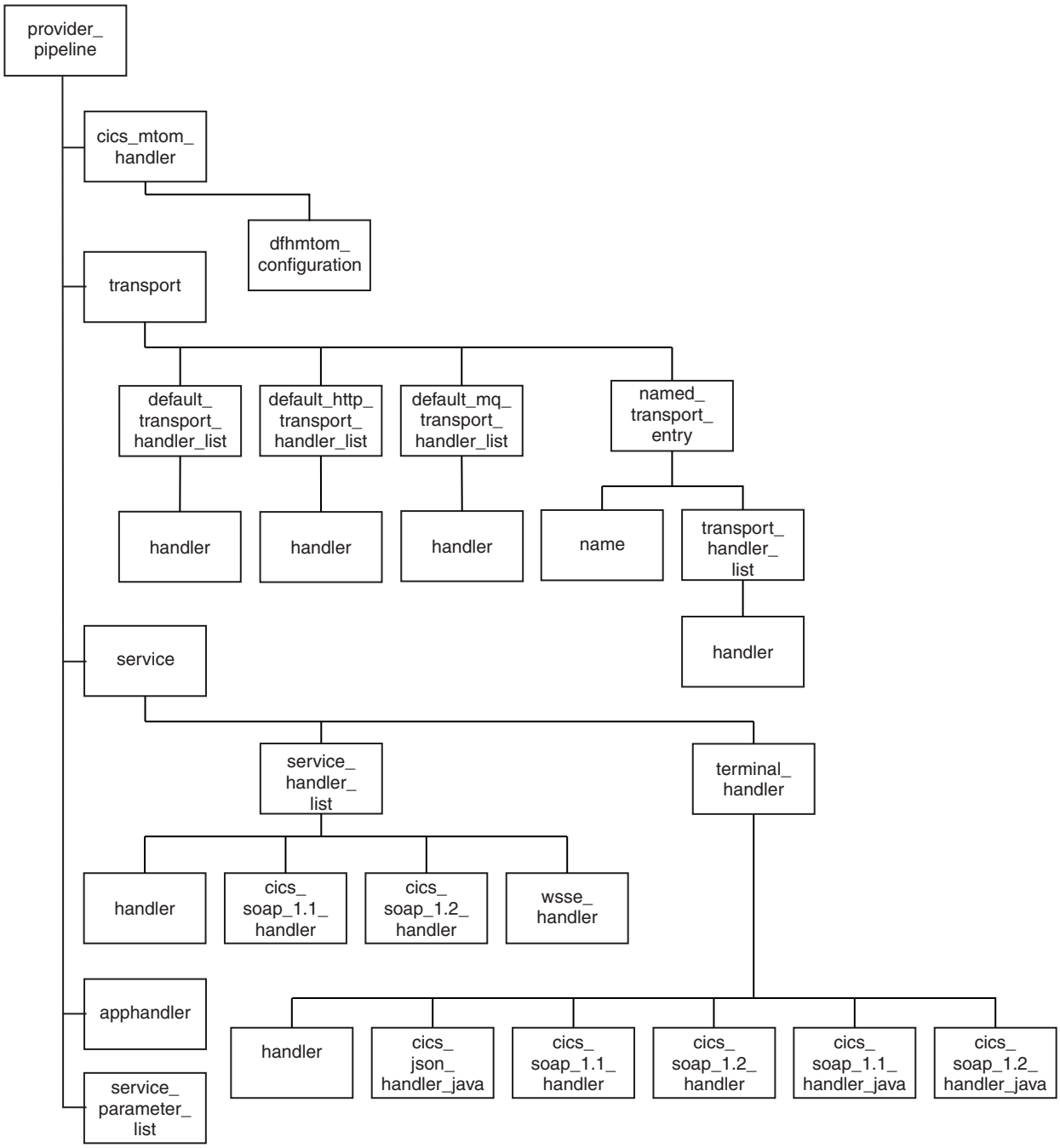


図 *20.* サービス・プロバイダーのパイプライン定義の構造：

注**:** 図を簡略化するために、<handler>、<cics_json_handler_java>、<cics_soap_1.1_handler>、<cics_soap_1.2_handler>、<cics_soap_1.1_handler_java>、<cics_soap_1.2_handler_java> の各エレメントの子エレメントは示されていません。

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、z/OS UNIX 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、<requester_pipeline> エレメントです。 この文書の上位構造を図 21 に示します。

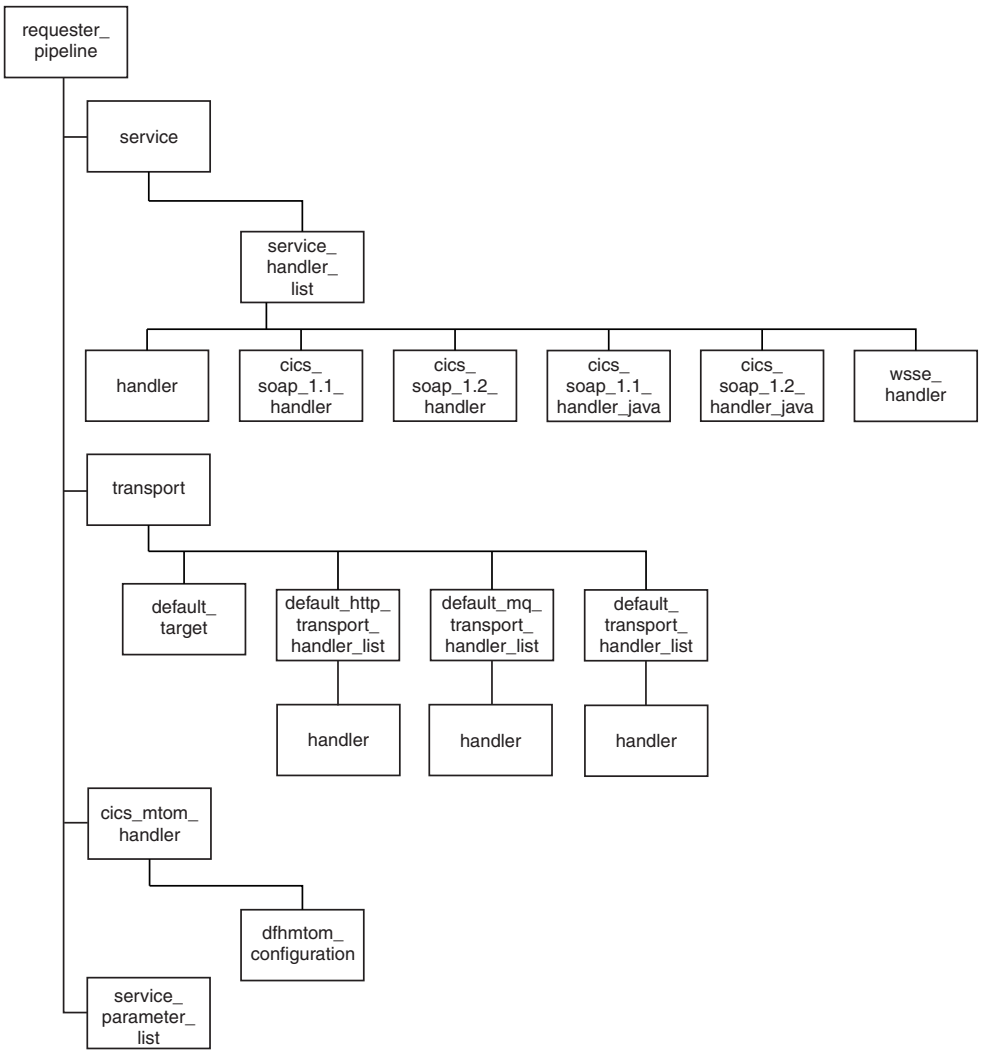


図 *21.* サービス・リクエスターのパイプライン定義の構造 :

注**:** 図を簡略化するために、<handler>、<cics_soap_1.1_handler>、<cics_soap_1.2_handler>、<cics_soap_1.1_handler_java>、および <cics_soap_1.2_handler_java> エレメントの子エレメントは示されていません。

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。

アプリケーション・ハンドラー**:**

アプリケーション・ハンドラーは、実行時に SOAP サービス・プロバイダー・パイプラインの端末ハンドラーがリンクする CICS プログラムです。

アプリケーション・ハンドラーは、提供されている SOAP メッセージ・ハンドラーのいずれかが端末ハンドラーになるようなプロバイダー・モード・パイプラインで使用されます。この状態は、<terminal_handler> エレメントに <cics_soap_1.1_handler>、<cics_soap_1.2_handler>、<cics_soap_1.1_handler_java>、または <cics_soap_1.2_handler_java> エレメントが含まれている場合に生じます。

アプリケーション・ハンドラーは、SOAP 要求の本体の処理および戻されたデータを使用した応答の生成を受け持ちます。 アプリケーション・ハンドラーは他のプログラムを呼び出して、この処理を完了することができます。 アプリケーション・ハンドラーは通常、1 つ以上のビジネス・アプリケーションの周りの汎用のプレゼンテーション層の役割を果たします。 これは、XML をアプリケーションが使用できる形式にマップし、そのアプリケーションに接続して、戻されたデータを使用して応答を生成する役割を受け持ちます。

CICS からアプリケーション・ハンドラーに接続する方法は 2 つあります。 典型的なメカニズムでは、チャネルおよび制御コンテナーを使用します。もう 1 つの方式では、Axis2 用 Java バインディングを使用します。

チャネル接続アプリケーション・ハンドラーは、<provider_pipeline> エレメントの <apphandler> エレメントで指定されます。 実行時に、DFHWS-APPHANDLER コンテナーに <apphandler> の内容が入ります。 しかし、DFHWS-APPHANDLER コンテナーは、他のメッセージ・ハンドラーによって動的に更新される場合があります。 そのため、実行時にリンクされるプログラムは、<apphandler> エレメントで指定されるプログラムとは異なる場合があります。以下のアプリケーション・ハンドラーは、<apphandler> エレメントまたは DFHWS-APPHANDLER コンテナーで指定できます。

- 提供されているチャネル接続 SOAP アプリケーション・ハンドラーである、DFHPITP。チャネル接続アプリケーション・ハンドラーについて詳しくは、161 ページの『チャネル接続のアプリケーション・ハンドラー』を参照してください。
- 独自のチャネル接続アプリケーション・ハンドラー。 このアプリケーション・ハンドラーは、Java 以外の言語で作成できます。 チャネル接続アプリケーション・ハンドラーで使用できる制御コンテナーについて詳しくは、 178 ページの『制御コンテナー』を参照してください。
- ApplicationHandler Java インターフェースを実装し、Axis2 MessageContext を使用してパイプラインに接続する、Java ベースのパイプライン用の独自の Java アプリケーション・ハンドラー。 ApplicationHandler Java インターフェースについて詳しくは、インターフェース ApplicationHandlerを参照してください。

Axis2 用 Java バインディングを使用するアプリケーション・ハンドラーを使用するには、<provider_pipeline> エレメントの <apphandler_class> エレメントを指定する必要があります。 また、Axis2 アプリケーション・ハンドラーを使用する場

合は、Web サービス・パイプラインとアプリケーション・ハンドラーを実行する JVM サーバーが存在し、Web サービス・パイプラインの終端ハンドラーが、メッセージ・ハンドラー <cics_soap_1.1_handler_java> または <cics_soap_1.2_handler_java> である必要があります。提供されている Axis2 アプリケーション・ハンドラーを使用するには、<apphandler_class> エレメントで com.ibm.cicsts.axis2.CICSAxis2ApplicationHandler を指定する必要がありますが、独自の Axis2 アプリケーション・ハンドラー・クラスを指定できます。 実行時に、DFHWS-APPHANCLAS コンテナーに <apphandler_class> の内容が入ります。

CICS Web サービス・アシスタントを使用して配置される Web サービス・アプリケーションでは、DFHPITP を指定するか、DFHPITP を使用する独自のアプリケーション・ハンドラーを <apphandler> エレメントに指定するか、<apphandler_class> エレメントに com.ibm.cicsts.axis2.CICSAxis2ApplicationHandler を指定する必要があります。CICS Web サービス・アシスタントについて詳しくは、CICS Web サービス・アシスタントを参照してください。

Web サービス配置の Axis2 スタイルを使用して、プロバイダー・モードの Web サービスとして CICS 内に Axis2 アプリケーションを配置することもできます。詳しくは、Axis2 JVM サーバーにおける Java プロバイダー・モードの Web サービスの配置を参照してください。

<apphandler_class> パイプライン構成エレメント**:**

パイプラインの端末ハンドラーが Axis2 アプリケーション・ハンドラーにリンクすることを指定します。

<terminal_handler> エレメントに <cics_json_handler_java>、<cics_soap_1.1_handler_java>、<cics_soap_1.2_handler_java> のいずれかのエレメントが含まれる場合に Axis2 アプリケーション・ハンドラーを指定するために、<apphandler_class> エレメントが使用されます。提供されている Axis2 アプリケーション・ハンドラーを使用するには、<apphandler_class> エレメントで com.ibm.cicsts.axis2.CICSAxis2ApplicationHandler を指定します。また、CICS SOAP ハンドラーを使用する場合には、独自の Axis2 アプリケーション・ハンドラー・クラスを指定することもできます。

または、チャネル接続アプリケーション・ハンドラーを使用する場合は、パイプライン構成ファイル内で、<apphandler> エレメントを指定することができます。詳しくは、<apphandler> エレメントを参照してください。ただし、<apphandler_class> エレメントと <apphandler> エレメントを同一のパイプライン構成ファイル内で指定することはできません。

注**:** <cics_json_handler_java> エレメントと一緒には <apphandler> エレメントを使用しないでください。

<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場合、<apphandler_class> エレメントを使用することはできません。

アプリケーション・ハンドラーについて詳しくは、 113 ページの『アプリケーション・ハンドラー』を参照してください。

使用先

- サービス・プロバイダー

格納元

- <provider_pipeline> エレメント

例

```
<apphandler_class>com.ibm.cicsts.axis2.CICSAxis2ApplicationHandler</apphandler_class>
```

<named_transport_entry> パイプライン構成エレメント**:**

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。

- WebSphere 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**:**

Name	説明
type	指定のリソースと関連したトランスポート機構 **wmq** 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い場合、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 type="http">
  <name>PORT80</name>
  <transport_handler_list>
    <handler><program>HANDLER1</program><handler_parameter_list/></handler>
    <handler><program>HANDLER2</program><handler_parameter_list/></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> パイプライン構成エレメント**:**

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・エレメントを指定します。

使用先

- サービス・プロバイダー

内容

1. オプションの <cics_mtom_handler> エレメント
2. オプションの <transport> エレメント
3. <service> エレメント
4. オプションの <apphandler> エレメント
5. オプションの <apphandler_class> エレメント
6. 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>
```

<provider_pipeline_json> パイプライン構成エレメント**:**

z/OS Connect の 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・エレメントを指定します。

これは、ハンドラー・プログラムを定義できないという点で <provider_pipeline> エレメントと異なります。パイプラインのこのスタイルは、z/OS Connect によって使用される **WEBSERVICE** リソースのコンテナーとして使用されます。 z/OS Connect 以外のものを使用して <provider_pipeline_json> パイプラインを開始しようとしても、エラーが発生します。結果として生じる **PIPELINE** リソースは、**USAGE(PIPELINE) URIMAP** リソースのターゲットとして使用できません。それは **USAGE(JVMSERVER) URIMAP** リソースでのみ使用できます。

使用先

- サービス・プロバイダー

属性**:**

java_parser={**yes**|**no**}
インバウンド・メッセージを処理するために使用される JSON パーサーのタイプを選択します。

可能な値は以下のとおりです。

yes JVM サーバー内の Java を使用して JSON 構文解析を実行します。これはデフォルトです。

no JSON メッセージの非 Java 構文解析を実行します。

注**:** java_parser 属性はオプションです。これを指定しない場合のデフォルト動作は、JVM サーバー内の Java を使用して JSON メッセージを構文解析することです。これは java_parser="yes" を指定するのと同じです。

java_generator={**yes**|**no**}

アウトバウンド・メッセージを生成するために使用する JSON 生成プログラムのタイプを選択します。

可能な値は以下のとおりです。

yes JVM サーバー内の Java を使用して JSON の生成を実行します。

no Java 以外での JSON メッセージの生成を実行します。これはデフォルトです。

内容

- <jvmserver> エレメント。 z/OS Connect が構成されている JVMSERVER リソースの名前が格納されています。

Java 構文解析を使用する例

```
<provider_pipeline_json java_parser="yes">
  <jvmserver>DFHWLP</jvmserver>
</provider_pipeline_json>
```

非 **Java** 構文解析を使用する例

```
<provider_pipeline_json java_parser="no">
  <jvmserver>DFHWLP</jvmserver>
</provider_pipeline_json>
```

<terminal_handler> パイプライン構成エレメント**:**

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_json_handler_java>

<cics_soap_1.1_handler>

<cics_soap_1.2_handler>

<cics_soap_1.1_handler_java>

<cics_soap_1.2_handler_java>

パイプラインで SOAP 1.1 と SOAP 1.2 の両方のメッセージを処理する場合、エレメント <cics_soap_1.2_handler> または <cics_soap_1.2_handler_java> を使用する必要があります。

要確認: サービス・プロバイダーでは、<service_handler_list> エレメントおよび <terminal_handler> エレメントで、<cics_soap_1.1_handler> および <cics_soap_1.2_handler> を指定できます。 ただしサービス・プロバイダーでは、<cics_soap_1.1_handler_java> および <cics_soap_1.2_handler_java> を指定できるのは、<terminal_handler> エレメント内に限られます。

例

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

例: **JSON** メッセージの非 **Java** 処理の使用可能化

JSON メッセージの非 Java 処理を使用可能にするには、次のように、端末ハンドラー・プログラムを DFHPIJT として指定します。

```
<terminal_handler>
  <handler>
      <program>DFHPIJT</program><handler_parameter_list/>
  </handler>
</terminal_handler>
```

注: DFHPIJT を端末ハンドラーとして使用する場合は、パイプライン構成ファイルにアプリケーション・ハンドラーを定義しないでください。つまり、パイプライン構成ファイルに <apphandler> エレメントが含まれていてはなりません。アプリケーション・ハンドラーを指定しても、呼び出されません。

<transport_handler_list> パイプライン構成エレメント:

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、ローカル入力キューの名前になります。
- 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> 構成エレメント**:**

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 <cics_mtom_handler> エレメント
4. 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>
  <service>
    <service_handler_list>
      <cics_soap_1.1_handler/>
    </service_handler_list>
  </service>
</requester_pipeline>
```

サービス・プロバイダーおよびサービス・リクエスターのパイプライン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。

<addressing> パイプライン構成エレメント**:**

Java ベースの SOAP 処理で Web Services Addressing のサポートを指定します。

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

<cics_soap_1.1_handler_java> エレメント

<cics_soap_1.2_handler_java> エレメント

内容

<namespace> エレメント。 サービス・プロバイダーでは、このエレメントはオプションです。 このエレメントには、CICS でサポートされている 2 つの WS-Addressing スキーマのいずれかが格納されます。 インバウンド・メッセージでは、Axis2 は両方の仕様をサポートします。 アウトバウンド・メッセージでは、このエレメントで指定されているネーム・スペースが使用されます。 このエレメントを指定しない場合、または 2 つのエレメントがある場合、CICS はインバウンド・メッセージと同じ仕様をアウトバウンド・メッセージで使用します。 サービス・リクエスターでは、このエレメントは必須であり、アウトバウンド・メッセージに名前空間を 1 つだけ指定できます。

この例では、サービス・プロバイダー・パイプラインの構成が示されています。ここでは、両方の WS-Addressing 仕様がサポートされています。 CICS はインバウンド・メッセージと同じ仕様をアウトバウンド・メッセージで使用します。 空の <addressing> エレメントを指定することにより、同じ結果が得られます。

```
<addressing>
       <namespace>http://www.w3.org/2005/08/addressing</namespace>
       <namespace>http://schemas.xmlsoap.org/ws/2004/08/addressing</namespace>
</addressing>
```

<cics_json_handler_java> パイプライン構成エレメント**:**

Java ベースの JSON パイプラインでの JSON メッセージのためにハンドラー・プログラムの属性を指定します。

使用先

- 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

1. <jvmserver> エレメント。
2. オプションの <repository> エレメント。

例

以下の例では、Java ベースの JSON ハンドラーとそのネストされているエレメントの XML が示されています。

```
<?xml version="1.0" encoding="EBCDIC-CP-US"?>
<provider_pipeline xmlns="http://www.ibm.com/software/htp/cics/pipeline">
  <service>
    <terminal_handler>
      <cics_json_handler_java>
        <jvmserver>DFHAXIS</jvmserver>
        <repository>/usr/lpp/cicsts/cicsts55/lib/pipeline/repository</repository>
      </cics_json_handler_java>
```

```
    </terminal_handler>
  </service>
  <apphandler_class>com.ibm.cicsts.axis2.CICSAxis2ApplicationHandler</apphandler_class>
</provider_pipeline>
```

<cics_soap_1.1_handle> パイプライン構成エレメント**:**

非 Java パイプラインでの SOAP 1.1 メッセージのためにハンドラー・プログラムの属性を指定します。

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. <program_name> 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 (Uniform 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" ...> .... </t: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> エレメントにアスタリスク (*) 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。 以下に例を示します。

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

<localname> エレメントにアスタリスクを指定すると、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、次のすべての <headerprogram> エレメントと一致します。

```
<headerprogram>
  <program_name>HDRPROG1</program_name>
  <namespace>http://mynamespace</namespace>
  <localname>*</localname>
  <mandatory>false</mandatory>
</headerprogram>
<headerprogram>
  <program_name>HDRPROG2</program_name>
  <namespace>http://mynamespace</namespace>
  <localname>myhead*</localname>
  <mandatory>false</mandatory>
</headerprogram>
<headerprogram>
  <program_name>HDRPROG3</program_name>
  <namespace>http://mynamespace</namespace>
  <localname>myheaderblock</localname>
  <mandatory>false</mandatory>
</headerprogram>
```

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 または false) が格納されています。代わりにそれぞれ 1 または 0 の値をコーディングできます。

   true

   サービス・プロバイダー・パイプラインでのサービス要求の処理中、およびサービス・リクエスター・パイプラインでのサービス応答の処理中に、SOAP メッセージに <namespace> および <localname> 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。

   サービス・リクエスター・パイプラインでのサービス要求の処理中、およびサービス・プロバイダー・パイプラインでのサービス応答の処理中に、CICS が作成する SOAP メッセージに最初に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。 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
SOAP メッセージに <namespace> および <localname> 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>
    <program_name> ... </program_name>
    <namespace>...</namespace>
    <localname>...</localname>
    <mandatory>true</mandatory>
  </headerprogram>
</cics_soap_1.1_handler>
```

<cics_soap_1.1_handler_java> パイプライン構成エレメント**:**

Java ベースの SOAP パイプラインでの SOAP 1.1 メッセージのためにハンドラー・プログラムの属性を指定します。

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

1. <jvmserver> エレメント。
2. オプションの <repository> エレメント。
3. オプションの <addressing> エレメント。 Axis2 で Web Services Addressing を使用可能にする場合、DFHWSADH ヘッダー処理プログラムを使用しないでください。
4. 存在しないか 1 つ以上の <headerprogram> エレメント。各 <headerprogram> エレメントの内容は、以下のとおりです。
   a. <program_name> 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。 Java で Axis2 ハンドラーを作成し、SOAP ヘッダーを処理できます。
   b. <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 (Uniform Resource Identifier) が格納されます。

c. <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" ...> .... </t:myheaderblock>
```

ネーム・スペース名は http://mynamespace、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> エレメントにアスタリスク (*) 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。 以下に例を示します。

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

<localname> エレメントにアスタリスクを指定すると、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、以下のヘッダー・ブロック

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

は、次のすべての <headerprogram> エレメントと一致します。

```
<headerprogram>
  <program_name>HDRPROG1</program_name>
  <namespace>http://mynamespace</namespace>
  <localname>*</localname>
  <mandatory>false</mandatory>
</headerprogram>
<headerprogram>
  <program_name>HDRPROG2</program_name>
  <namespace>http://mynamespace</namespace>
  <localname>myhead*</localname>
  <mandatory>false</mandatory>
</headerprogram>
<headerprogram>
  <program_name>HDRPROG3</program_name>
  <namespace>http://mynamespace</namespace>
  <localname>myheaderblock</localname>
  <mandatory>false</mandatory>
</headerprogram>
```

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> エレメントと <localname> エレメントは同じであるが、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。

d. <mandatory> エレメント。XML ブール値 (true または false) が格納されています。代わりにそれぞれ 1 または 0 の値をコーディングできます。

true

サービス・プロバイダー・パイプラインでのサービス要求の処理中、およびサービス・リクエスター・パイプラインでのサービス応答の処理中に、SOAP メッセージに <namespace> および <localname> 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。

サービス・リクエスター・パイプラインでのサービス要求の処理中、およびサービス・プロバイダー・パイプラインでのサービス応答の処理中に、 CICS が作成する SOAP メッセージに最初に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。 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

SOAP メッセージに <namespace> および <localname> 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

以下の例では、Java ベースの SOAP ハンドラーとそのネストされているエレメントの XML が示されています。

```
<cics_soap_1.1_handler_java>
  <jvmserver>JVMSERV1</jvmserver>
  <headerprogram>
    <program_name>HDRPROG4</program_name>
    <namespace>http://mynamespace</namespace>
```

```
    <localname>myheaderblock</localname>
    <mandatory>true</mandatory>
  </headerprogram>
</cics_soap_1.1_handler_java>
```

<cics_soap_1.2_handler> パイプライン構成エレメント**:**

非 Java パイプラインでの SOAP 1.2 メッセージのためにハンドラー・プログラムの属性を指定します。

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. <program_name> 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 (Uniform 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" ...> .... </t: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> エレメントにアスタリスク (*) 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。 以下に例を示します。

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

<localname> エレメントにアスタリスクを指定すると、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、次のすべての <headerprogram> エレメントと一致します。

```
<headerprogram>
  <program_name>HDRPROG1</program_name>
  <namespace>http://mynamespace</namespace>
  <localname>*</localname>
  <mandatory>false</mandatory>
</headerprogram>
<headerprogram>
  <program_name>HDRPROG2</program_name>
  <namespace>http://mynamespace</namespace>
  <localname>myhead*</localname>
  <mandatory>false</mandatory>
</headerprogram>
<headerprogram>
  <program_name>HDRPROG3</program_name>
  <namespace>http://mynamespace</namespace>
  <localname>myheaderblock</localname>
  <mandatory>false</mandatory>
</headerprogram>
```

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 または false) が格納されています。代わりにそれぞれ 1 または 0 の値をコーディングできます。

   true

   サービス・プロバイダー・パイプラインでのサービス要求の処理中、およびサービス・リクエスター・パイプラインでのサービス応答の処理中に、SOAP メッセージに <namespace> および <localname> 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。

   サービス・リクエスター・パイプラインでのサービス要求の処理中、およびサービス・プロバイダー・パイプラインでのサービス応答の処理中に、CICS が作成する SOAP メッセージに最初に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。 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
SOAP メッセージに <namespace> および <localname> 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>
    <program_name> ... </program_name>
    <namespace>...</namespace>
    <localname>...</localname>
    <mandatory>true</mandatory>
  </headerprogram>
</cics_soap_1.2_handler>
```

<cics_soap_1.2_handler_java> パイプライン構成エレメント**:**

Java ベースの SOAP パイプラインでの SOAP 1.2 メッセージのためにハンドラー・プログラムの属性を指定します。

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

1. <jvmserver> エレメント。
2. オプションの <repository> エレメント。
3. オプションの <addressing> エレメント。 Axis2 で Web Services Addressing のサポートを使用可能にする場合、ヘッダー処理プログラムを使用しないでください。 Java で Axis2 ハンドラーを作成し、SOAP ヘッダーを処理できます。
4. 存在しないか 1 つ以上の <headerprogram> エレメント。各 <headerprogram> エレメントの内容は、以下のとおりです。
   a. <program_name> 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。
   b. <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 (Uniform Resource Identifier) が格納されます。

c. <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" ...> .... </t:myheaderblock>
```

ネーム・スペース名は http://mynamespace、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> エレメントにアスタリスク (*) 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。 以下に例を示します。

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

<localname> エレメントにアスタリスクを指定すると、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、以下のヘッダー・ブロック

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

は、次のすべての <headerprogram> エレメントと一致します。

```
<headerprogram>
  <program_name>HDRPROG1</program_name>
  <namespace>http://mynamespace</namespace>
  <localname>*</localname>
  <mandatory>false</mandatory>
</headerprogram>
<headerprogram>
  <program_name>HDRPROG2</program_name>
  <namespace>http://mynamespace</namespace>
  <localname>myhead*</localname>
  <mandatory>false</mandatory>
</headerprogram>
<headerprogram>
  <program_name>HDRPROG3</program_name>
  <namespace>http://mynamespace</namespace>
  <localname>myheaderblock</localname>
  <mandatory>false</mandatory>
</headerprogram>
```

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> エレメントと <localname> エレメントは同じであるが、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。

d. <mandatory> エレメント。XML ブール値 (true または false) が格納されています。代わりにそれぞれ 1 または 0 の値をコーディングできます。

true

サービス・プロバイダー・パイプラインでのサービス要求の処理中、およびサービス・リクエスター・パイプラインでのサービス応答の処理中に、SOAP メッセージに <namespace> および <localname> 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。

サービス・リクエスター・パイプラインでのサービス要求の処理中、およびサービス・プロバイダー・パイプラインでのサービス応答の処理中に、 CICS が作成する SOAP メッセージに最初に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。 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

SOAP メッセージに <namespace> および <localname> 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

以下の例では、Java ベースの SOAP ハンドラーとそのネストされているエレメントの XML が示されています。

```
<cics_soap_1.2_handler_java>
  <jvmserver>JVMSERV1</jvmserver>
  <headerprogram>
    <program_name>HDRPROG4</program_name>
    <namespace>http://mynamespace</namespace>
```

```
    <localname>myheaderblock</localname>
    <mandatory>true</mandatory>
  </headerprogram>
</cics_soap_1.2_handler_java>
```

<default_http_transport_handler_list> パイプライン構成エレメント**:**

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リストがあまり具体的でない場合のみです。

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> パイプライン構成エレメント**:**

WebSphere 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。

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リストがあまり具体的でない場合のみです。

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> パイプライン構成エレメント**:**

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。

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ハンドラーのリストがあまり具体的でない場合に、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>
     <program>HANDLER1</program>
     <handler_parameter_list/>
  </handler>
  <handler>
     <program>HANDLER2</program>
     <handler_parameter_list/>
  </handler>
</default_transport_handler_list>
```

<handler> パイプライン構成エレメント**:**

メッセージ・ハンドラー・プログラムの属性を指定します。

CICS 提供のハンドラー・プログラムの中には、<handler> エレメントを使用しないものがあります。例えば、CICS 提供の SOAP メッセージ・ハンドラー・プログラムは、<cics_soap_1.1_handler> エレメント、<cics_soap_1.2_handler> エレメント、<cics_soap_1.1_handler_java> エレメント、および <cics_soap_1.2_handler_java> 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>

<default_http_transport_handler_list>

<default_mq_transport_handler_list>

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

```
<?xml version="1.0"?>
<provider_pipeline>
        xmlns="http://www.ibm.com/software/htp/cics/pipeline">
 <service>
  <service_handler_list>
   <handler>
     <program>MYPROG</program>
     <handler_parameter_list><output print="yes"/></handler_parameter_list>
   </handler>
  </service_handler_list>
  <terminal_handler>
   <cics_soap_1.1_handler>
    ...
   </cics_soap_1.1_handler>
  </terminal_handler>
 </service
 <apphandler>DFHPITP</apphandler>
</provider_pipeline>
```

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。

<jvmserver> パイプライン構成エレメント**:**

JVMSERVER リソースの名前を指定します。

このエレメントは、要求を処理する JVMSERVER リソースの名前を識別します。値が指定されない場合、エラー・メッセージが生成され、PIPELINE が DISABLED 状態でインストールされます。

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

- <cics_json_handler_java> エレメント
- <cics_soap_1.1_handler_java> エレメント
- <cics_soap_1.2_handler_java> エレメント
- <provider_pipeline_json> エレメント

例

```
<jvmserver>JVMSERVER_NAME</jvmserver>
```

<repository> パイプライン構成エレメント**:**

Axis2 リポジトリーのディレクトリー名を指定します。

このオプション・エレメントは、Axis2 リポジトリーのディレクトリー名を指定します。 このオプションを使用する場合、事前にハンドラー XML で <jvmserver> エレメント を指定する必要があります。エレメントが提供されない場合は、サンプル・リポジトリーが使用されます。 CICS Transaction Server をインストールすると、サンプルの Axis2 リポジトリーが `/usr/lpp/cicsts/cicsts55/lib/pipeline/repository` ディレクトリーにインストールされます (`/usr/lpp/cicsts/cicsts55` は、z/OS UNIX の CICS ファイルのデフォルト・インストール・ディレクトリーです)。

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

- <cics_json_handler_java> エレメント
- <cics_soap_1.1_handler_java> エレメント
- <cics_soap_1.2_handler_java> エレメント

例

```
<cics_soap_1.1_handler_java>
 <jvmserver>JVMSERV1</jvmserver>
 <repository>/lib/pipeline/repository</repository>
</cics_soap_1.1_handler_java>
```

<service> パイプライン構成エレメント**:**

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> パイプライン構成エレメント**:**

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

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

- <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

<cics_soap_1.1_handler>

<cics_soap_1.2_handler>

<cics_soap_1.1_handler_java>

<cics_soap_1.2_handler_java>

<handler>

<wsse_handler>

<service_handler_list> エレメントでハンドラー・エレメントを指定する順序で、各ハンドラーが実行時に呼び出される順序を決定します。 例えば、パイプラインが WS-Security をサポートしている場合、暗号化された SOAP メッセージは、<wsse_handler> エレメントが呼び出されるまで、暗号化されたままです。 したがって、暗号化されていないメッセージを処理する他のどのハンドラー・プログラムよりも前に、<wsse_handler> エレメントを指定する必要があります。

<cics_soap_1.1_handler_java> および <cics_soap_1.2_handler_java> エレメントは、Java ベース・パイプラインの <terminal_handler> エレメントで指定する必要があるため、サービス・プロバイダーでは、それらのエレメントを <service_handler_list> エレメントに含めることはできません。サービス・リクエスターには、<cics_soap_1.1_handler_java> および

<cics_soap_1.2_handler_java> を含めることができますが、それらのエレメントを使用する場合、それらを <service_handler_list> エレメント内でリストされる最初のエレメントにする必要があります。

パイプラインで SOAP 1.1 と SOAP 1.2 の両方のメッセージを処理する場合、エレメント <cics_soap_1.2_handler> または <cics_soap_1.2_handler_java> を使用する必要があります。

サービス・リクエスター・パイプラインでは SOAP 1.1 または SOAP 1.2 のいずれかのハンドラーを使用できますが、この場合 SOAP 1.2 ハンドラーは SOAP 1.1 メッセージをサポートしません。サービス・リクエスター・アプリケーションが DFHREQUEST コンテナー内の完全な SOAP エンベロープを送信する場合は、パイプラインに SOAP 1.1 または SOAP 1.2 ハンドラーを指定しないでください。これを行うと、アウトバウンド・メッセージで SOAP メッセージ・ヘッダーが重複するのを回避できます。

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、汎用ハンドラーと SOAP ハンドラーを <terminal_handler> エレメントに指定できます。SOAP ヘッダーの処理について詳しくは、 171 ページの『ヘッダー処理プログラム』を参照してください。

例

```
<service_handler_list>
  <wsse_handler>
    ...
  </wsse_handler>
 <cics_soap_1.1_handler_java>
    ...
 </cics_soap_1.1_handler_java>
 <handler>
    ...
 </handler>
</service_handler_list>
```

<service_parameter_list> パイプライン構成エレメント**:**

パイプライン内のすべてのメッセージ・ハンドラーで使用できる XML エレメントをコンテナー DFH-SERVICEPLIST で指定します。これはオプションのエレメントです。

使用先

- サービス・リクエスター
- サービス・プロバイダー

内容

- WS-AT を使用している場合は、1 つの <registration_service_endpoint> エレメント。
- サービス・リクエスターで WS-AT を使用している場合、オプションの <new_tx_context_required/> エレメント
- オプションのユーザー定義タグ

例

```
<requester_pipeline>
 <service_parameter_list>
  <registration_service_endpoint>
  http://provider.example.com:7160/cicswsat/RegistrationService
  </registration_service_endpoint>
  <new_tx_context_required/>
  <user_defined_tag1>
  ...
  </user_defined_tag1>
 </service_parameter_list>
</requester_pipeline>
```

<transport> パイプライン構成エレメント**:**

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。ただし、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サービス・プロバイダー・アプリケーションは、通常は Web サービス記述から URI 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>
```

MTOM/XOP に合わせたパイプライン構成

CICS SOAP パイプラインは、Message Transmission Optimization Mechanism (MTOM) 仕様および XML-binary Optimized Packaging (XOP) 仕様をサポートできます。 これらの仕様では、SOAP を使用して、Base64 エンコードのオーバーヘッドなしでバイナリー・データを送受信するためのメカニズムが定義されています。 MTOM サポートを有効にするには、そのサポートに合わせてパイプラインを構成する必要があります。

<mtom> パイプライン構成エレメント**:**

Java ベースのパイプラインの MTOM/XOP サポートを有効にします。 このエレメントをパイプライン構成ファイルで定義すると、すべてのインバウンド・メッセージとアウトバウンド・メッセージで MTOM サポートが有効になります。 一方、このエレメントをパイプライン構成ファイルで指定しない場合は、インバウンド・メッセージだけで MTOM サポートが有効になります。

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

<cics_soap_1.1_handler_java>

<cics_soap_1.2_handler_java>

プロバイダー側とリクエスター側の両方のパイプライン構成ファイルで、オプションの <addressing> エレメントとオプションの <headerprogram> エレメントの間に、<mtom> エレメントを定義する必要があります。

例

プロバイダー・モードまたはリクエスター・モードのパイプラインで、以下のように指定できます。

```
<cics_soap_1.2_handler_java>
  <jvmserver>JVMSERV1</jvmserver>
  <addressing></addressing>
  <mtom></mtom>
  <headerprogram>
    <program_name>HDRPROG4</program_name>
    <namespace>http://mynamespace</namespace>
    <localname>myheaderblock</localname>
    <mandatory>true</mandatory>
  </headerprogram>
</cics_soap_1.2_handler_java>
```

<cics_mtom_handler> パイプライン構成エレメント**:**

SOAP パイプラインの場合に、用意されている MTOM ハンドラー・プログラムを有効にします。 このプログラムは、XOP 文書とバイナリー添付ファイルが含まれている MTOM MIME multipart/related メッセージをサポートします。 MTOM サポートは、パイプライン内に受信されるすべてのインバウンド・メッセージについて使用可能になりますが、アウトバウンド・メッセージについての MTOM サポートは、追加のオプションに従って条件付きで使用可能になります。

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>

プロバイダー・パイプライン構成ファイルでは、<cics_mtom_handler> エレメントを <transport> エレメントの前に定義してください。実行時に、トランスポート・ハンドラーを含む他のハンドラーが処理する前に、MTOM ハンドラー・プログラムがインバウンド MTOM メッセージをアンパックする必要があります。 その後、応答メッセージの最終ハンドラーとして呼び出され、MTOM メッセージをパックして Web サービス・リクエスターに送信します。

リクエスター・パイプライン構成ファイルでは、<cics_mtom_handler> エレメントを <transport> エレメントの後に定義してください。実行時に、他のすべてのハンドラーが処理するまで、アウトバウンド要求メッセージは MTOM フォーマットに変換されません。 この後、インバウンド応答メッセージの最初のハンドラーとして呼び出され、他のハンドラーが処理する前に MTOM メッセージをアンパックし、要求しているプログラムに戻します。

注**:** Java ベースのパイプラインでは、このハンドラー・プログラムを使用できません。 Java ベースのパイプラインでは、<mtom> エレメントを指定してください。

内容

<dfhmtom_configuration> エレメント

デフォルト・オプションは、<dfhmtom_configuration> エレメントに指定される構成オプションを使用して変更できます。デフォルト・オプションを変更しない場合は、空のエレメントを使用できます。

例

プロバイダー・モードのパイプラインでは、以下のように指定できます。

```
<provider_pipeline>
    <cics_mtom_handler></cics_mtom_handler>
    <transport>
    ....
    </transport>
```

```
    <service>
    ....
    </service>
</provider_pipeline>
```

<dfhmtom_configuration> パイプライン構成エレメント**:**

Java をサポートしないパイプラインの場合に、用意されている MTOM ハンドラー・プログラムの構成情報を指定します。 このプログラムは、XOP 文書とバイナリー添付ファイルが含まれている MIME メッセージをサポートします。 MTOM についての構成を何も指定しないと、CICS はデフォルト値を想定します。

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

```
<cics_mtom_handler>
```

属性**:**

Name	説明
version	構成情報のバージョンを示す整数。 有効な値は 1 のみです。

内容

- オプションの <mtom_options> エレメント
- オプションの <xop_options> エレメント
- オプションの <mime_options> エレメント

例

```
<dfhmtom_configuration version="1">
  <mtom_options send_mtom="same" send_when_no_xop="no"/>
  <xop_options apphandler_supports_xop="yes"/>
  <mime_options content_id_domain="example.org"/>
</dfhmtom_configuration>
```

<mtom_options> パイプライン構成エレメント**:**

Java をサポートしないパイプラインの場合に、アウトバウンド SOAP メッセージで MTOM を使用するときの条件を指定します。

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

```
<dfhmtom_configuration>
```

属性**:**

属性	説明
send_mtom	アウトバウンド SOAP メッセージを MIME メッセージに変換する場合に MTOM を使用するかどうかを指定します。 **no** アウトバウンド SOAP メッセージに MTOM を使用しません。 **same** サービス・プロバイダー・モードでは、リクエスターが MTOM を使用するときはいつでも、SOAP 応答メッセージに MTOM を使用します。 これは、サービス・プロバイダー・パイプラインでのデフォルト値です。 サービス・リクエスター・モードでは、この値を指定することは、`send_mtom="yes"` を指定することと同じです。 **yes** すべてのアウトバウンド SOAP メッセージに MTOM を使用します。 これは、サービス・リクエスター・パイプラインでのデフォルト値です。
send_when_no_xop	メッセージにバイナリー添付ファイルが存在しない場合でも MTOM メッセージを送信するかどうかを指定します。 **no** メッセージとともにバイナリー添付ファイルを送信する場合のみ、MTOM を使用します。 **yes** メッセージに送信するバイナリー添付ファイルが存在しない場合でも、すべてのアウトバウンド SOAP メッセージに MTOM を使用します。 これはデフォルト値で、主として送信側が MTOM/XOP をサポートする受信側プログラムへの標識として使用されます。 この属性は、任意の **send_mtom** 属性値と組み合わせることができますが、`send_mtom="no"` を指定する場合は無効になります。

例

```
<provider_pipeline>
 <cics_mtom_handler>
  <dfhmtom_configuration version="1">
    <mtom_options send_mtom="same" send_when_no_xop="no"/>
  </dfhmtom_configuration>
 </cics_mtom_handler>
...
</provider_pipeline>
```

このプロバイダー・パイプラインの例では、メッセージとともにバイナリー添付ファイルを送信する必要があり、サービス・リクエスターが MTOM メッセージを送信した場合のみ、SOAP メッセージは MTOM メッセージに変換されます。

<xop_options> パイプライン構成エレメント**:**

Java をサポートしないパイプラインで、XOP 処理を直接モードで実行するか、互換モードで実行するかを指定します。

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

```
<dfhmtom_configuration>
```

属性**:**

属性	説明
apphandler_ supports_xop	プロバイダー・モードでは、アプリケーション・ハンドラーが直接モードで XOP 文書を処理できる場合に指定します。 **no** アプリケーション・ハンドラーは XOP 文書を直接処理できません。これは、<apphandler> エレメントで DFHPITP が指定されない場合のデフォルト値です。 互換モードは、MTOM 形式で受信あるいは送信されたインバウンド・メッセージまたはアウトバウンド・メッセージを処理するためにパイプラインで使用されます。 **yes** アプリケーション・ハンドラーは XOP 文書を処理できます。 これは、<apphandler> エレメントで DFHPITP が指定された場合のデフォルト値です。 直接モードは、MTOM 形式で受信あるいは送信されたインバウンド・メッセージまたはアウトバウンド・メッセージを処理するためにパイプラインで使用されます。 これは、実行時に制約の対象となります。 例えば、パイプライン構成ファイルで WS-Security 関連のエレメントを指定した場合、MTOM ハンドラーは、パイプラインでは XOP 文書の処理に直接モードではなく互換モードを使用することを決定します。 リクエスター・モードでは、サービス・リクエスター・アプリケーションが CICS Web サービス・サポートを使用して、直接モードで XOP 文書を作成および処理する場合に指定します。 **no** サービス・リクエスター・アプリケーションは、CICS Web サービス・サポートを使用しません。この値は、リクエスター・アプリケーションがパイプラインを実行するため DFHPIRT にリンクしているために、XOP 文書を直接モードで作成および処理できない場合に指定します。 **yes** サービス・リクエスター・アプリケーションは、CICS Web サービス・サポートを使用します。この値は、リクエスター・アプリケーションが **EXEC CICS INVOKE WEBSERVICE** コマンドを使用する場合に指定します。

例

```
<provider_pipeline>
 <cics_mtom_handler>
  <dfhmtom_configuration version="1">
    <xop_options apphandler_supports_xop="no"/>
  </dfhmtom_configuration>
 </cics_mtom_handler>
 ...
</provider_pipeline>
```

このプロバイダー・パイプラインの例では、インバウンド MTOM メッセージとアウトバウンド応答メッセージは、互換モードを使用してパイプラインで処理されます。

<mime_options> パイプライン構成エレメント**:**

Java をサポートしないパイプラインの場合に、MIME コンテンツ ID 値の生成時に使用するドメイン名を指定します。 MIME コンテンツ ID 値は、バイナリー添付ファイルを識別するために使用されます。

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

```
<dfhmtom_configuration>
```

属性**:**

属性	説明
content_id_domain	使用する構文は *domain.name* です。 インターネット標準に準拠するためには、名前は有効なインターネット・ホスト名で、パイプラインがインストールされている CICS システムに対して一意でなければなりません。 これは CICS によって検査されないことに注意してください。 このエレメントを省略すると、CICS は値 `cicsts` を使用します。

例

```
<provider_pipeline>
<dfhmtom_configuration version="1">
  <mime_options content_id_domain="example.org"/>
</dfhmtom_configuration>
...
</provider_pipeline>
```

この例では、バイナリー添付ファイルへの参照は、`cid:`*`unique_value`*`@example.org` を使用して作成されます。

WS-Security に合わせたパイプライン構成

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-Security プロトコルに参加するには、それに応じてパイプラインを構成する必要があります。このためには、メッセージ・ハンドラー DFHWSSE を組み込んで、ハンドラーの構成情報を提供します。

例

以下は、WS-Security を使用するプロバイダー・パイプライン構成ファイルの形式の一例です。

```
<?xml version="1.0"?>
<provider_pipeline
         xmlns="http://www.ibm.com/software/htp/cics/pipeline">
   <service>
      <service_handler_list>
         <wsse_handler>
            <dfhwsse_configuration version="1">
               <authentication trust="blind" mode="basic"/>
            </dfhwsse_configuration>
         </wsse_handler>
         <handler>
          ...
         </handler>
      </service_handler_list>
      <terminal_handler>
         <cics_soap_1.2_handler/>
     </terminal_handler>
   </service>
   <apphandler>DFHPITP</apphandler>
</provider_pipeline>
```

<wsse_handler> パイプライン構成エレメント**:**

WS-Security のサポートを提供する CICS 提供のメッセージ・ハンドラーで使用されるパラメーターを指定します。

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

```
<service_handler_list>
```

内容

- <dfhwsse_configuration> パイプライン構成エレメント。

プロバイダー・パイプライン構成ファイル内で、WS-Security 用の CICS 提供メッセージ・ハンドラーが暗号化メッセージを暗号化解除する必要が生じることがあります。暗号化されていないメッセージ内容を処理する必要のある他のすべてのハンドラー・プログラムの前に、<wsse_handler> エレメントを定義する必要があります。

リクエスター・パイプライン構成ファイル内で、WS-Security 用の CICS 提供メッセージ・ハンドラーがメッセージを暗号化する必要が生じることがあります。暗号化されていないメッセージ内容を処理する必要のある他のすべてのハンドラー・プログラム (CICS SOAP ハンドラー・プログラムを含む) の後に、これを定義する必要があります。

<dfhwsse_configuration> パイプライン構成エレメント**:**

Web サービスの保護についてのサポートを提供する、セキュリティー・ハンドラー DFHWSSE1 の構成情報を指定します。

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

<wsse_handler>

属性:

Name	説明
version	構成情報のバージョンを示す整数。 有効な値は 1 のみです。

内容

1. 以下のいずれかのエレメント
   - オプションの <authentication> エレメント。
     - サービス・リクエスター・パイプラインでは、<authentication> エレメントが、アウトバウンド SOAP メッセージのセキュリティー・ヘッダーで使用する必要がある認証のタイプを指定します。
     - サービス・プロバイダー・パイプラインでは、このエレメントが、CICS がインバウンド SOAP メッセージでセキュリティー・トークンを使用して、処理が行われるユーザー ID を決定するかどうかを指定します。
   - オプションの <sts_authentication> エレメント。

     このエレメントの action 属性は、Security Token Service に送信する要求のタイプを指定します。 要求が ID トークンの発行である場合、CICS は、ネストされたエレメント内の値を使用して、指定されたタイプの ID トークンを要求します。
2. <sts_authentication> エレメントを指定する場合は、 <sts_endpoint> エレメントも指定する必要があります。

   このエレメントが存在するとき、CICS は、<endpoint> エレメント内の URI を使用して、要求を Security Token Service に送信します。
3. オプションの、空の <expect_signed_body/> エレメント。

   <expect_signed_body/> エレメントは、インバウンド・メッセージの <body> に署名が必要であることを示します。 インバウンド・メッセージの本文が正しく署名されていない場合、CICS はセキュリティー障害でメッセージを拒否します。
4. オプションの、空の <expect_encrypted_body/> エレメント。

   <expect_encrypted_body/> > エレメントは、インバウンド・メッセージの <body> を暗号化する必要があることを示します。インバウンド・メッセージの本文が正しく暗号化されていない場合、CICS はセキュリティー障害でメッセージを拒否します。
5. オプションの <sign_body> エレメント。

このエレメントが存在する場合、CICS は、<sign_body> エレメント内に含まれる <algorithm> エレメントに指定されるアルゴリズムを使用して、アウトバウンド・メッセージの <body> に署名します。

6. オプションの <encrypt_body> エレメント。

   このエレメントが存在する場合、CICS は、<encrypt_body> エレメント内に含まれ、<algorithm> エレメントに指定されるアルゴリズムを使用して、アウトバウンド・メッセージの <body> を暗号化します。

7. プロバイダー・パイプラインのみで、オプションの <reject_signature/> エレメント。

   このエレメントが存在する場合、CICS は、メッセージの本文の一部またはすべてを署名する証明書がヘッダーに含まれているメッセージを拒否します。 Web サービス・リクエスターに SOAP 障害が発行されます。

8. プロバイダー・パイプラインのみで、オプションの <reject_encryption> エレメント。

   このエレメントが存在する場合、CICS は、部分的または完全に暗号化されているメッセージを拒否します。 Web サービス・リクエスターに SOAP 障害が発行されます。

例

```
<dfhwsse_configuration version="1">
  <sts_authentication action="issue">
    <auth_token_type>
      <namespace>http://example.org.tokens</namespace>
      <element>UsernameToken</element>
    </auth_token_type>
    <suppress/>
  </sts_authentication>
  <sts_endpoint>
    <endpoint>https://example.com/SecurityTokenService</endpoint>
  </sts_endpoint>
  <expect_signed_body/>
  <expect_encrypted_body/>
  <sign_body>
    <algorithm>http://www.w3.org/2000/09/xmldsig#rsa-sha1</algorithm>
    <certificate_label>SIGCERT01</certificate_label>
  </sign_body>
  <encrypt_body>
    <algorithm>http://www.w3.org/2001/04/xmlenc#tripledes-cbc</algorithm>
    <certificate_label>ENCCERT02</certificate_label>
  </encrypt_body>
</dfhwsse_configuration>
```

<authentication> パイプライン構成エレメント**:**

インバウンドおよびアウトバウンド SOAP メッセージのヘッダー内の、セキュリティー・トークンの使用について指定します。

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

```
<dfhwsse_configuration>
```

属性:

属性	説明
trust および mode	trust 属性と mode 属性を一緒に使用して、次のことを指定します。 • 宣言 ID が使用されるかどうか • SOAP メッセージで使用されるセキュリティー・トークンの組み合わせ 宣言 ID によって、信頼できるユーザーがその ID に関連する資格情報を持っていなくても、別の ID、つまり宣言 *ID* の下で作業を実行することを、信頼できるユーザーが宣言できます。 宣言 ID が使用されるとき、メッセージには *trust* トークン および *ID* トークン が含まれます。 trust トークンは、宣言 ID に対する正しい許可が送信側にあるか検査するために使用されます。ID トークンには、宣言 ID、つまり要求が実行されるユーザー ID が格納されます。 宣言 ID を使用するには、サービス・プロバイダーがリクエスターを信頼してこの宣言を行う必要があります。CICS では、信頼関係は、セキュリティー・マネージャーの代理定義で確立されます。要求している ID には、宣言 ID の代わりに作業を開始するための、正しい権限が必要です。 これらの属性の有効な組み合わせとそれぞれの意味を表 3 と 149 ページの表 4 にまとめておきます。

表 3. サービス・リクエスター・パイプラインにおける **mode** および **trust** 属性

trust	**mode**	意味
なし	なし	メッセージに資格情報が追加されません
なし	basic	属性値の組み合わせが無効です
なし	signature	宣言 ID は使用されません。 CICS は、メッセージに追加され、メッセージの本文の署名に使用される、単一の X.509 セキュリティー・トークンを使用します。証明書は `<certificate_label>` エレメントで識別され、アルゴリズムは `<algorithm>` エレメントで指定されます。
blind	なし	属性値の組み合わせが無効です
blind	basic	宣言 ID は使用されません。 CICS は ID トークンをメッセージに追加しますが、trust トークンは提供しません。ID トークンは、パスワードのないユーザー名です。ID トークンに配置されるユーザー ID は、DFHWS-USERID コンテナーの内容です (デフォルトで、実行中のタスクのユーザー ID を含みます)。
blind	signature	属性値の組み合わせが無効です
basic	なし	属性値の組み合わせが無効です
basic	basic	属性値の組み合わせが無効です

表 3. サービス・リクエスター・パイプラインにおける **mode** および **trust** 属性 (続き)

trust	**mode**	意味
basic	signature	属性値の組み合わせが無効です
signature	なし	属性値の組み合わせが無効です
signature	basic	宣言 ID が使用されます。 CICS はメッセージに以下のトークンを追加します。 • trust トークンは、X.509 セキュリティー・トークンです。 • ID トークンは、パスワードのないユーザー名です。 ID トークンおよびメッセージの本文の署名に使用される証明書は、<certificate_label> によって指定されます。 ID トークンに配置されるユーザー ID は、DFHWS-USERID コンテナーの内容です (デフォルトで、実行中のタスクのユーザー ID を含みます)。
signature	signature	属性値の組み合わせが無効です

表 4. サービス・プロバイダー・パイプラインにおける **mode** および **trust** 属性

trust	**mode**	意味
なし	なし	インバウンド・メッセージに資格情報を組み込む必要はありません。CICS は、メッセージ内の資格情報の抽出や検証を実行しようとはしません。ただし、CICS は署名された要素が正しく署名されているかどうかをチェックします。
なし	basic	インバウンド・メッセージに、パスワードのあるユーザー名セキュリティー・トークンが含まれている必要があります。CICS はそのユーザー名を DFHWS-USERID コンテナーに入れます。
なし	basic-ICRX	属性値の組み合わせが無効です
なし	basic-kerberos	属性値の組み合わせが無効です
なし	signature	インバウンド・メッセージに、メッセージの本文の署名に使用された X.509 セキュリティー・トークンが含まれる必要があります。
blind	なし	属性値の組み合わせが無効です
blind	basic	インバウンド・メッセージに、ID トークンが含まれる必要があります。 ID トークンには、ユーザー ID と、オプションでパスワードが含まれます。 CICS はそのユーザー ID を DFHWS-USERID コンテナーに入れます。パスワードが含まれない場合は、CICS はユーザー ID を検証せずに使用します。パスワードが含まれる場合は、セキュリティー・ハンドラー DFHWSSE1 が検証します。

表 4. サービス・プロバイダー・パイプラインにおける **mode** および **trust** 属性 (続き)

trust	**mode**	意味
blind	basic-ICRX	インバウンド・メッセージには、ICRX ID トークンが含まれている必要があります。 CICS は ID を解決し、ユーザー ID を DFHWS-USERID コンテナーに入れ、ICRX を DFHWS-ICRX コンテナーに入れます。認証 (必要な場合) では、クライアント認証 SSL または別のセキュリティー・プロトコルが使用されます。
blind	basic-kerberos	属性値の組み合わせが無効です
blind	signature	インバウンド・メッセージに、ID トークンが含まれる必要があります。 ID トークンは、SOAP メッセージ・ヘッダー内の最初の X.509 証明書です。 この証明書でメッセージに署名をしておく必要はありません。セキュリティー・ハンドラーが一致するユーザー ID を抽出し、DFHWS-USERID コンテナーに配置します。
basic	なし	属性値の組み合わせが無効です
basic	basic	インバウンド・メッセージは宣言 ID を使用する必要があります。 • trust トークンは、パスワードのあるユーザー名トークンです。 • ID トークンは、パスワードのない 2 番目のユーザー名トークンです。CICS はそのユーザー名をコンテナー DFHWS-USERID に入れます。
basic	basic-ICRX	インバウンド・メッセージは宣言 ID を使用する必要があります。 • trust トークンは、パスワードのあるユーザー名トークンです。 CICS はユーザー ID とパスワードの組み合わせが有効かどうかを確認し、有効であれば、CICS は ICRX ベースの宣言 ID をユーザー ID に解決します。その後 CICS は、認証 ID から宣言 ID への代理セキュリティー検査を実行します。 • ID トークンは ICRX です。これは、特定のクライアント・ユーザーを識別します。 CICS はユーザー名をコンテナー DFHWS-USERID に入れ、ICRX をコンテナー DFHWS-ICRX に入れます。

表 4. サービス・プロバイダー・パイプラインにおける **mode** および **trust** 属性 (続き)

trust	**mode**	意味
basic	basic-kerberos	インバウンド・メッセージは宣言 ID を使用する必要があります。 1 つのトークンは必須で、それは以下のフォーマット・タイプのいずれかの Kerberos バージョン 5 トークンです: • http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#Kerberosv5_AP_REQ1510 • http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#GSS_Kerberosv5_AP_REQ1510 • http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#Kerberosv5_AP_REQ4120 • http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#GSS_Kerberosv5_AP_REQ4120 このトークンは Base-64 でエンコードされている必要があります。CICS は Network Authentication Service for z/OS を使用してトークンを検証し、トークンに関連付けられているユーザー ID をコンテナーに入れます。
basic	signature	インバウンド・メッセージは宣言 ID を使用する必要があります。 • trust トークンは、パスワードのあるユーザー名トークンです。 • ID トークンは、X.509 証明書です。 CICS は、証明書に関連したユーザー ID をコンテナー DFHWS-USERID に入れます。
signature	なし	属性値の組み合わせが無効です
signature	basic	インバウンド・メッセージは宣言 ID を使用する必要があります。 • trust トークンは X.509 証明書です • ID トークンは、パスワードのないユーザー名トークンです。 CICS は、そのユーザー名をコンテナー DFHWS-USERID に入れます。 ID トークンおよび本文は、X.509 証明書で署名する必要があります。

表 4. サービス・プロバイダー・パイプラインにおける **mode** および **trust** 属性 (続き)

trust	**mode**	意味
signature	basic-ICRX	インバウンド・メッセージは宣言 ID を使用する必要があります。 • trust トークンは X.509 証明書で署名された ICRX です。 CICS は X.509 証明書をユーザー ID に解決し、XML Signature が有効かどうかを確認します。CICS は ICRX ベースの宣言 ID をユーザー ID に解決します。その後 CICS は、認証 X.509 ID から宣言 ICRX ID への代理セキュリティー検査を実行します。 • ID トークンは、パスワードのないユーザー名トークンです。 CICS はユーザー名をコンテナー DFHWS-USERID に入れ、ICRX をコンテナー DFHWS-ICRX に入れます。
signature	basic-kerberos	属性値の組み合わせが無効です
signature	signature	インバウンド・メッセージは宣言 ID を使用する必要があります。 • trust トークンは X.509 証明書です • ID トークンは、2 番目の X.509 証明書です。CICS は、この証明書に関連したユーザー ID をコンテナー DFHWS-USERID に入れます。 ID トークンおよび本文は、最初の X.509 証明書 (trust トークン) で署名する必要があります。

注:

1. trust 属性値と mode 属性値の組み合わせは、PIPELINE がインストールされるときに検査されます。 属性が正しくコーディングされていない場合は、インストールが失敗します。
2. CICS は、VERIFY PHRASEで説明されているプロセスの中で、パスワード検証を使用してユーザー ID を検証します。

内容

1. オプションの、空の `<suppress/>` エレメント。

   このエレメントがサービス・プロバイダー・パイプラインに指定される場合、ハンドラーは、作業が行われるユーザー ID を決定するメッセージ内のどのセキュリティー・トークンの使用も試みません。

   このエレメントがサービス・リクエスター・パイプラインに指定される場合、ハンドラーは、アウトバウンド SOAP メッセージに、認証に必要な、どのセキュリティー・トークンの追加も試みません。
2. リクエスター・パイプラインでは、SOAP メッセージの本文の署名に使用されるアルゴリズムの URI を指定する、オプションの `<algorithm>` エレメント。trust 属性と mode 属性の値の組み合わせが、メッセージが署名されていることを示している場合は、このエレメントを指定する必要があります。 このエレ

メントでは、SHA1 を使用する RSA アルゴリズムだけを指定できます。 URI は http://www.w3.org/2000/09/xmldsig#rsa-sha1 です。

3. オプションとして、<certificate_label> エレメントを組み込めます。このエレメントでは、RACF® にインストールされている X.509 デジタル証明書に関連付けるラベルを指定できます。このエレメントがサービス・リクエスター・パイプラインに指定され、<suppress> エレメントが指定されない場合は、証明書が SOAP メッセージのセキュリティー・ヘッダーに追加されます。<certificate_label> エレメントを指定しない場合は、 CICS が RACF 鍵リングでデフォルトの証明書を使用します。

   このエレメントはサービス・プロバイダー・パイプラインでは無視されます。

例

```
<authentication trust="signature" mode="basic">
  <suppress/>
  <algorithm>http://www.w3.org/2000/09/xmldsig#rsa-sha1</algorithm>
  <certificate_label>AUTHCERT03</certificate_label>
</authentication>
```

<sts_authentication> パイプライン構成エレメント:

認証のために Security Token Service (STS) を使用する必要があることを指定し、送信される要求のタイプを決定します。

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

<dfhwsse_configuration>

属性:

Name	説明
アクション	サービス・プロバイダー・パイプラインでメッセージを受信したときに CICS が STS に送信する要求のタイプを指定します。 有効な値は、以下のとおりです。 **issue** STS は、SOAP メッセージの識別トークンを発行します。プロバイダー・パイプライン内の SAML に対しては、この値は無効です。 **validate** STS は、提供された識別トークンを妥当性検査して、トークンが有効かどうかをセキュリティー・ハンドラーに戻します。 この属性を指定しない場合、CICS は、アクションが ID トークンを要求することであると想定します。 サービス・リクエスター・パイプラインでは、CICS は必ず STS によるトークンの発行を要求するため、この属性を指定することはできません。

Name	説明
extract	この属性は、SAML を使用する場合にのみ有効です。SAML トークンのエレメントが抽出されるかどうかを指定します。有効な値は、以下のとおりです。 **no** SAML トークンのエレメントはコンテナーの中に抽出されません。 **yes** SAML トークンの主なエレメントが抽出されて、CICS 作成のコンテナーにそれらが入れられます。
token_signature	この属性は、SAML を使用する場合にのみ有効です。トークン・シグニチャーを提供する必要があるかどうかを指定します。有効な値は、以下のとおりです。 **ignored** 提供されるシグニチャーはすべて無視されます。 **required** 有効なシグニチャーを提供する必要があります。これはデフォルト値です。

Name	説明
tran_channel	この属性は、SAML を使用する場合にのみ有効です。サービス・プロバイダー・パイプラインの中で、この属性は、パイプラインで受信されるメッセージに含まれる SAML アサーションを、トランザクション・チャネル DFHTRANSACTION のコンテナー内でターゲット・アプリケーション・プログラムが使用できるようにするかどうかを指定します。有効な値は、以下のとおりです。 **yes** DFHTRANSACTION チャネルのコンテナーの中に SAML アサーションがコピーされ、プログラムで使用可能になります。コンテナーの名前とタイプについて詳しくは、SAML のリンク可能インターフェース、DFHSAMLを参照してください。 **no** プログラムは DFHTRANSACTION チャネルを介して SAML アサーションを使用することはできませんが、パイプラインからプログラムに渡されるチャネルのコンテナー内では使用できます。これはデフォルト値です。 サービス・プロバイダーに対してこの属性を指定しない場合、SOAP パイプラインからプログラムに渡されるチャネルのコンテナー内でのみ、アサーションが使用可能になります。 サービス・リクエスター・パイプラインの中で、この属性は、トランザクション・チャネル DFHTRANSACTION の DFHSAML-OUTTOKEN コンテナーに含まれる SAML トークンが要求で使用されるかどうかを指定します。有効な値は、以下のとおりです。 **yes** DFHTRANSACTION チャネルの DFHSAML-OUTTOKEN コンテナーの内容が、要求の SAML トークンとして使用されます。 **no** パイプラインに渡されるチャネル内の DFHSAML-OUTTOKEN コンテナーの内容が、要求の SAML トークンとして使用されます。これはデフォルト値です。 サービス・リクエスターに対してこの属性を指定しない場合、SOAP パイプラインに渡されるチャネル内の DFHSAML-OUTTOKEN コンテナーから SAML トークンが取られます。

内容

1. <auth_token_type> エレメント。 このエレメントは、サービス・リクエスター・パイプラインで <sts_authentication> エレメントを指定する場合は必須で、サービス・プロバイダー・パイプラインではオプションです。詳しくは、<auth_token_type> エレメントを参照してください。
   - サービス・リクエスター・パイプラインでは、<auth_token_type> エレメントは、CICS が DFHWS-USERID コンテナーに含まれるユーザー ID を

STS に送信したときに、STS が発行するトークンのタイプを示します。 CICS が STS から受け取るトークンは、アウトバウンド・メッセージのヘッダーに置かれます。

- サービス・プロバイダー・パイプラインでは、<auth_token_type> エレメントは、CICS がメッセージ・ヘッダーから取得して、交換または妥当性検査のために STS に送信する識別トークンを判別するために使用されます。 CICS は最初に、メッセージ・ヘッダーで指定されたタイプの識別トークンを使用します。 このエレメントを指定しない場合、CICS は、メッセージ・ヘッダーで見つけた最初の識別トークンを使用します。 CICS は、次のものは ID トークンと見なしません。
  - wsu:Timestamp
  - xenc:ReferenceList
  - xenc:EncryptedKey
  - ds:Signature

2. サービス・プロバイダー・パイプラインの場合に限り、オプションの空の <suppress/> エレメント。このエレメントが指定される場合、ハンドラーは、操作の実行に使われるユーザー ID を判別する目的でメッセージ内のどのセキュリティー・トークンの使用も試みません。 <suppress/> エレメントは、STS によって戻される ID トークンが含まれます。

例

次の例は、セキュリティー・ハンドラーが STS からトークンを要求するサービス・プロバイダー・パイプラインを示します。

```
<sts_authentication action="issue">
   <auth_token_type>
      <namespace>http://example.org.tokens</namespace>
      <element>UsernameToken</element>
   </auth_token_type>
   <suppress/>
</sts_authentication>
```

<auth_token_type> パイプライン構成エレメント:

必要な ID トークンのタイプを指定します。

このエレメントは、サービス・リクエスターで <sts_authentication> エレメントを指定するときは必須で、サービス・プロバイダーではオプションです。

- サービス・リクエスター・パイプラインでは、<auth_token_type> エレメントは、CICS が DFHWS-USERID コンテナーに含まれるユーザー ID を STS に送信したときに、STS が発行するトークンのタイプを示します。 CICS が STS から受け取るトークンは、アウトバウンド・メッセージのヘッダーに置かれます。
- サービス・プロバイダー・パイプラインでは、<auth_token_type> エレメントは、CICS がメッセージ・ヘッダーから取得して、交換または妥当性検査のために STS に送信する識別トークンを判別するために使用されます。 CICS は最初に、メッセージ・ヘッダーで指定されたタイプの識別トークンを使用します。 このエレメントを指定しない場合、CICS は、メッセージ・ヘッダーで見つけた最初の識別トークンを使用します。 CICS は、次のものは ID トークンと見なしません。

- wsu:Timestamp
- xenc:ReferenceList
- xenc:EncryptedKey
- ds:Signature

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

<sts_authentication>

内容

1. <namespace> エレメント。 このエレメントには、検証または交換の対象となるトークン・タイプのネーム・スペースが含まれます。

   SAML を使用する場合は、SAML のバージョンに応じて、このエレメントの内容を urn:oasis:names:tc:SAML:1.0:assertion または urn:oasis:names:tc:SAML:2.0:assertion のいずれかに設定します。
2. <element> エレメント。 このエレメントには、検証または交換の対象となるトークン・タイプのローカル名が含まれます。

   SAML の場合、ローカル名 Assertion を使用してください。

これらのエレメントの値は、トークンの Qname を形成します。

例

```
<auth_token_type>
   <namespace>http://example.org.tokens</namespace>
   <element>UsernameToken</element>
</auth_token_type>
```

<sts_endpoint> パイプライン構成エレメント**:**

Security Token Service (STS) の場所を指定します。

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

<dfhwsse_configuration>

内容

- <endpoint> エレメント。 このエレメントには、ネットワーク上の Security Token Service (STS) の場所を指し示す URI が含まれます。 STS への接続を安全に保つには、HTTP ではなく、SSL または TLS を使用することをお勧めします。

SAML サポートを使用するには、エンドポイントを cics://PROGRAM/DFHSAML に設定します。

また、WebSphere MQ エンドポイントは、JMS フォーマットの URI を使用して指定することもできます。

- オプションの <jvmserver> エレメント。このエレメントは、SAML トークン・サービスを実行するよう構成された JVM サーバーを識別します。このエレメントが含まれていない場合、デフォルトのサンプル・リソース JVM サーバー DFHXSTS が想定されます。このエレメントは、SAML を使用する場合にのみ有効です。その他の状況でこれを使用した場合、エラーが発生します。

例

この例では、指定した URI にある STS へのセキュア接続を使用するようエンドポイントが構成されています。

```
<sts_endpoint>
   <endpoint>https://example.com/SecurityTokenService</endpoint>
</sts_endpoint>
```

この例では、CICS SAML サポートを使用するようエンドポイントが構成されています。

```
<sts_endpoint>
   <endpoint>cics://PROGRAM/DFHSAML</endpoint>
</sts_endpoint>
```

<sign_body> パイプライン構成エレメント:

アウトバウンド SOAP メッセージの本文に署名するよう DFHWSSE に指示して、メッセージに署名する方法に関する情報を提供します。

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

```
<dfhwsse_configuration>
```

内容

1. SOAP メッセージの本文に署名するために使用されるアルゴリズムを特定する URI を含む <algorithm> エレメント。 署名アルゴリズムに示したアルゴリズムを指定できます。
2. RACF にインストールされたデジタル証明書に関連付けられたラベルを指定する <certificate_label> エレメント。デジタル証明書は、メッセージへの署名に使用される鍵を提供します。

例

```
<sign_body>
 <algorithm>http://www.w3.org/2000/09/xmldsig#rsa-sha1</algorithm>
 <certificate_label>SIGCERT01</certificate_label>
</sign_body>
```

<encrypt_body> パイプライン構成エレメント**:**

アウトバウンド SOAP メッセージの本文を暗号化するよう DFHWSSE に指示して、メッセージを暗号化する方法に関する情報を提供します。

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

```
<dfhwsse_configuration>
```

内容

1. SOAP メッセージの本文を暗号化するために使用される、アルゴリズムを特定する URI を含む <algorithm> エレメント。 暗号化アルゴリズムに示したアルゴリズムを指定できます。
2. RACF 内のデジタル証明書に関連付けられたラベルを指定する <certificate_label> エレメント。 デジタル証明書は、メッセージの暗号化に使用される鍵を提供します。

例

```
<encrypt_body>
  <algorithm>http://www.w3.org/2001/04/xmlenc#aes256-cbc</algorithm>
  <certificate_label>ENCCERT02</certificate_label>
</encrypt_body>
```

アプリケーション・ハンドラー

アプリケーション・ハンドラーは、実行時に SOAP サービス・プロバイダー・パイプラインの端末ハンドラーがリンクする CICS プログラムです。

アプリケーション・ハンドラーは、提供されている SOAP メッセージ・ハンドラーのいずれかが端末ハンドラーになるようなプロバイダー・モード・パイプラインで使用されます。この状態は、<terminal_handler> エレメントに <cics_soap_1.1_handler>、<cics_soap_1.2_handler>、<cics_soap_1.1_handler_java>、または <cics_soap_1.2_handler_java> エレメントが含まれている場合に生じます。

アプリケーション・ハンドラーは、SOAP 要求の本体の処理および戻されたデータを使用した応答の生成を受け持ちます。 アプリケーション・ハンドラーは他のプログラムを呼び出して、この処理を完了することができます。 アプリケーション・ハンドラーは通常、1 つ以上のビジネス・アプリケーションの周りの汎用のプレゼンテーション層の役割を果たします。 これは、XML をアプリケーションが使用できる形式にマップし、そのアプリケーションに接続して、戻されたデータを使用して応答を生成する役割を受け持ちます。

CICS からアプリケーション・ハンドラーに接続する方法は 2 つあります。 典型的なメカニズムでは、チャネルおよび制御コンテナーを使用します。もう 1 つの方式では、Axis2 用 Java バインディングを使用します。

チャネル接続アプリケーション・ハンドラーは、<provider_pipeline> エレメントの <apphandler> エレメントで指定されます。 実行時に、DFHWS-APPHANDLER コンテナーに <apphandler> の内容が入ります。 しかし、DFHWS-APPHANDLER コンテナーは、他のメッセージ・ハンドラーによって動的に更新される場合があります。 そのため、実行時にリンクされるプログラムは、<apphandler> エレメントで指定されるプログラムとは異なる場合があります。以下のアプリケーション・ハンドラーは、<apphandler> エレメントまたは DFHWS-APPHANDLER コンテナーで指定できます。

- 提供されているチャネル接続 SOAP アプリケーション・ハンドラーである、DFHPITP。チャネル接続アプリケーション・ハンドラーについて詳しくは、161 ページの『チャネル接続のアプリケーション・ハンドラー』を参照してください。
- 独自のチャネル接続アプリケーション・ハンドラー。 このアプリケーション・ハンドラーは、Java 以外の言語で作成できます。 チャネル接続アプリケーション・ハンドラーで使用できる制御コンテナーについて詳しくは、 178 ページの『制御コンテナー』を参照してください。
- ApplicationHandler Java インターフェースを実装し、Axis2 MessageContext を使用してパイプラインに接続する、Java ベースのパイプライン用の独自の Java アプリケーション・ハンドラー。 ApplicationHandler Java インターフェースについて詳しくは、インターフェース ApplicationHandlerを参照してください。

Axis2 用 Java バインディングを使用するアプリケーション・ハンドラーを使用するには、<provider_pipeline> エレメントの <apphandler_class> エレメントを指定する必要があります。 また、Axis2 アプリケーション・ハンドラーを使用する場合は、Web サービス・パイプラインとアプリケーション・ハンドラーを実行する JVM サーバーが存在し、Web サービス・パイプラインの終端ハンドラーが、メッセージ・ハンドラー <cics_soap_1.1_handler_java> または <cics_soap_1.2_handler_java> である必要があります。提供されている Axis2 アプリケーション・ハンドラーを使用するには、<apphandler_class> エレメントで com.ibm.cicsts.axis2.CICSAxis2ApplicationHandler を指定する必要がありますが、独自の Axis2 アプリケーション・ハンドラー・クラスを指定できます。 実行時に、DFHWS-APPHANCLAS コンテナーに <apphandler_class> の内容が入ります。

CICS Web サービス・アシスタントを使用して配置される Web サービス・アプリケーションでは、DFHPITP を指定するか、DFHPITP を使用する独自のアプリケーション・ハンドラーを <apphandler> エレメントに指定するか、<apphandler_class> エレメントに com.ibm.cicsts.axis2.CICSAxis2ApplicationHandler を指定する必要があります。CICS Web サービス・アシスタントについて詳しくは、CICS Web サービス・アシスタントを参照してください。

Web サービス配置の Axis2 スタイルを使用して、プロバイダー・モードの Web サービスとして CICS 内に Axis2 アプリケーションを配置することもできます。詳しくは、Axis2 JVM サーバーにおける Java プロバイダー・モードの Web サービスの配置を参照してください。

チャネル接続のアプリケーション・ハンドラー

チャネル接続のアプリケーション・ハンドラーは、チャネルおよび制御コンテナーを使用して CICS に接続されるアプリケーション・ハンドラーです。

アプリケーション・ハンドラーによって使用されるチャネルは、DFHAHC-V1 チャネルです。 このチャネルは、端末ハンドラーとプロバイダー・モードの Web サービス・アプリケーションの間で以下のコンテナーを渡します。

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、アプリケーションは内容を変更します。アプリケーションが内容を変更しない場合は、アプリケーション・ハンドラー・プログラムがこのコンテナーの内容を更新する必要があります。端末ハンドラーに戻る前に同じ内容をコンテナーに戻す場合でも、同様に更新する必要があります。

DFHNORESPONSE
サービス・リクエスター・パイプラインの要求段階では、サービス・プロバイダーは応答を戻さなくてもよいことを示します。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。

- コンテナー DFHNORESPONSE が存在する場合は、応答は見込まれません。
- コンテナー DFHNORESPONSE が不在の場合は、応答が見込まれます。

チャネルは、端末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。 例えば、ヘッダー処理プログラムはコンテナーをチャネルに追加できます。 これらのコンテナーは、ユーザー・コンテナーとして渡されます。 アプリケーション・ハンドラーについて詳しくは、 113 ページの『アプリケーション・ハンドラー』を参照してください。

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 または JSON 要求または応答の内容を、内容を変更せずに調べる
- XML または JSON 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 または JSON 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: SOAP メッセージを処理する場合、SOAP ハンドラー、<cics_soap_1.1_handler>、<cics_soap_1.2_handler>、<cics_soap_1.1_handler_java>、または <cics_soap_1.2_handler_java> を使用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。 コンテナーは次のタイプに分類できます。

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。 メッセージ・ハンドラーは、制御コンテナーを使用して後続のハンドラー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。

コンテナーがパイプライン・プロトコルを制御する方法

DFHFUNCTION、DFHREQUEST、および DFHRESPONSE コンテナーの内容が一緒にパイプライン・プロトコルを制御します。

パイプラインの実行の 2 つの段階 (要求段階と応答段階) で、DFHFUNCTION の値が、各メッセージ・ハンドラーに渡される制御コンテナーを決定します。

DFHFUNCTION	コンテキスト	DFHREQUEST	DFHRESPONSE
RECEIVE-REQUEST	サービス・プロバイダー、要求段階	存在 (長さ > 0)	存在 (長さ = 0)
SEND-RESPONSE	サービス・プロバイダー、応答段階	不在	存在 (長さ > 0)
SEND-REQUEST	サービス・リクエスター、要求段階	存在 (長さ > 0)	存在 (長さ = 0)
RECEIVE-RESPONSE	サービス・リクエスター、応答段階	不在	存在 (長さ > 0)
PROCESS-REQUEST	サービス・プロバイダー、端末ハンドラー	存在 (長さ > 0)	存在 (長さ = 0)
HANDLER-ERROR	サービス・リクエスターまたはサービス・プロバイダー、どちらかの段階	不在	存在 (長さ = 0)
NO-RESPONSE	サービス・リクエスターまたはサービス・プロバイダー、応答段階	不在	不在

その後の処理は、メッセージ・ハンドラーがパイプラインに戻すコンテナーによって決定されます。

要求段階中

- メッセージ・ハンドラーは DFHREQUEST コンテナーを戻すことができます。 処理は次のハンドラーを使用して要求段階で続行します。 コンテナー内のデータの長さをゼロにしてはなりません。
- メッセージ・ハンドラーは DFHRESPONSE コンテナーを戻すことができます。 処理は応答段階に切り替わり、DFHFUNCTION をサービス・プロバイダーで SEND-RESPONSE に、サービス・リクエスターで RECEIVE-RESPONSE に設定して、同じハンドラーが呼び出されます。 コンテナー内のデータの長さをゼロにしてはなりません。
- メッセージ・ハンドラーはコンテナーを戻すことができません。 処理は応答段階に切り替わり、DFHFUNCTION を NO-RESPONSE に設定して、同じハンドラーが呼び出されます。

端末ハンドラーで **(**サービス・プロバイダーのみ**)**

- メッセージ・ハンドラーは DFHRESPONSE コンテナーを戻すことができます。 処理は応答段階に切り替わり、DFHFUNCTION の新しい値 (SEND-RESPONSE) で、前のハンドラーが呼び出されます。 コンテナー内のデータの長さをゼロにしてはなりません。
- メッセージ・ハンドラーはコンテナーを戻すことができません。 処理は応答段階に切り替わり、DFHFUNCTION の新しい値 (NO-RESPONSE) で、前のハンドラーが呼び出されます。

応答段階中

- メッセージ・ハンドラーは DFHRESPONSE コンテナーを戻すことができます。 処理は応答段階で続行し、次のハンドラーが呼び出されます。コンテナー内のデータの長さをゼロにしてはなりません。
- メッセージ・ハンドラーはコンテナーを戻すことができません。 処理は応答段階で続行し、DFHFUNCTION の新しい値 (NO-RESPONSE) で、シーケンスの次のハンドラーが呼び出されます。

重要**:** 要求段階で、メッセージ・ハンドラーは DFHREQUEST または DFHRESPONSE を戻すことができますが、両方を戻すことはできません。メッセージ・ハンドラーが呼び出されるときに両方のコンテナーが存在しているので、どちらかを削除する必要があります。
次の表に、DFHFUNCTION のすべての値と、各メッセージ・ハンドラーによって戻される DFHREQUEST と DFHRESPONSE のすべての組み合わせについて、パイプラインによってとられるアクションを示します。

DFHFUNCTION	コンテキスト	**DFHREQUEST**	**DFHRESPONSE**	アクション
RECEIVE-REQUEST	サービス・プロバイダー、要求段階	存在 (長さ > 0)	存在	(エラー)
RECEIVE-REQUEST	サービス・プロバイダー、要求段階	存在 (長さ > 0)	不在	RECEIVE-REQUEST 関数で次のハンドラーを呼び出す
RECEIVE-REQUEST	サービス・プロバイダー、要求段階	存在 (長さ = 0)	適用外	(エラー)
RECEIVE-REQUEST	サービス・プロバイダー、要求段階	不在	存在 (長さ > 0)	応答段階に切り替え、SEND-RESPONSE 関数で同じハンドラーを呼び出す
RECEIVE-REQUEST	サービス・プロバイダー、要求段階	不在	存在 (長さ = 0)	(エラー)
RECEIVE-REQUEST	サービス・プロバイダー、要求段階	不在	不在	NO-RESPONSE 関数で同じハンドラーを呼び出す
SEND-RESPONSE	サービス・プロバイダー、応答段階	適用外	存在 (長さ > 0)	SEND-RESPONSE 関数で前のハンドラーを呼び出す
SEND-RESPONSE	サービス・プロバイダー、応答段階	適用外	存在 (長さ = 0)	(エラー)
SEND-RESPONSE	サービス・プロバイダー、応答段階	適用外	不在	NO-RESPONSE 関数で同じハンドラーを呼び出す
SEND-REQUEST	サービス・リクエスター、要求段階	存在 (長さ > 0)	存在 (長さ ≥ 0)	(エラー)
SEND-REQUEST	サービス・リクエスター、要求段階	存在 (長さ > 0)	不在	SEND-REQUEST 関数で次のハンドラーを呼び出す
SEND-REQUEST	サービス・リクエスター、要求段階	存在 (長さ = 0)	適用外	(エラー)
SEND-REQUEST	サービス・リクエスター、要求段階	不在	存在 (長さ > 0)	応答段階に切り替え、RECEIVE-RESPONSE 関数で前のハンドラーを呼び出す
SEND-REQUEST	サービス・リクエスター、要求段階	不在	存在 (長さ = 0)	(エラー)
SEND-REQUEST	サービス・リクエスター、要求段階	不在	不在	NO-RESPONSE 関数で同じハンドラーを呼び出す
RECEIVE-RESPONSE	サービス・リクエスター、応答段階	適用外	存在 (長さ > 0)	RECEIVE-RESPONSE 関数で前のハンドラーを呼び出す

DFHFUNCTION	コンテキスト	DFHREQUEST	DFHRESPONSE	アクション
RECEIVE-RESPONSE	サービス・リクエスター、応答段階	適用外	存在 (長さ = 0)	(エラー)
RECEIVE-RESPONSE	サービス・リクエスター、応答段階	適用外	不在	NO-RESPONSE 関数で同じハンドラーを呼び出す
PROCESS-REQUEST	サービス・プロバイダー、端末ハンドラー	適用外	存在 (長さ > 0)	RECEIVE-RESPONSE 関数で前のハンドラーを呼び出す
PROCESS-REQUEST	サービス・プロバイダー、端末ハンドラー	適用外	存在 (長さ = 0)	(エラー)
PROCESS-REQUEST	サービス・プロバイダー、端末ハンドラー	適用外	不在	NO-RESPONSE 関数で同じハンドラーを呼び出す
HANDLER-ERROR	サービス・リクエスターまたはサービス・プロバイダー、どちらかの段階	適用外	存在 (長さ > 0)	SEND-RESPONSE 関数または RECEIVE-RESPONSE 関数で前のハンドラーを呼び出す
HANDLER-ERROR	サービス・リクエスターまたはサービス・プロバイダー、どちらかの段階	適用外	存在 (長さ = 0)	(エラー)
HANDLER-ERROR	サービス・リクエスターまたはサービス・プロバイダー、どちらかの段階	適用外	不在	NO-RESPONSE 関数で同じハンドラーを呼び出す

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際、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。

このタスクについて

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 メッセージ・ハンドラーおよび 1.2 メッセージ・ハンドラーを使用できます。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。

手順

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. プログラムをコンパイルして、リンク・エディットします。 メッセージ・ハンドラー・プログラムは通常、属性 `TASKDATALOC(ANY)` で定義されるトランザクシ

ョン CPIH の下で実行されます。 したがって、プログラムをリンク・エディットする際は、AMODE(31) オプションを指定する必要があります。

3. 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。
4. 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。

このタスクについて

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。

手順

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・ハンドラーが渡されたコンテナーをまったく変更しない場合、エラーになります。

   メッセージ・ハンドラー・プログラムが以下のいずれかを戻すと、エラーとなります。
   - 空の DFHRESPONSE コンテナー。
   - 空ではない DFHREQUEST コンテナーおよび空ではない DFHRESPONSE コンテナー。
   - アウトバウンド要求での空の DFHREQUEST コンテナー。

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。

手順

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。

タスクの結果

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。

手順

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。

タスクの結果

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。

手順

1. コンテナー DFHREQUEST を削除します。
2. コンテナー 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、2 回目にメッセージ・ハンドラーを呼び出す前に CICS が応答を待機しないことを示すために、特別な対策を講じる必要があります。

このタスクについて

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

標準的な端末ハンドラーは、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。

このタスクについて

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。 標準的な端末ハンドラーでは、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。

手順

1. コンテナー DFHFUNCTION の内容を使用して、このハンドラーに渡されたメッセージが要求であることと、このハンドラー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	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
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、このハンドラーは端末ハンドラーではなく、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、端末ハンドラーはアプリケーション・プログラムを呼び出します。
4. 応答を作成して、コンテナー DFHRESPONSE に書き込みます。 応答がない場合は、コンテナー DFHRESPONSE を削除する必要があります。

タスクの結果

応答は、パイプラインの応答段階で次のハンドラーに渡されます。 ハンドラーは、機能 SEND-RESPONSE のために呼び出されます。応答がない場合は、次のハンドラーが機能 NO-RESPONSE のために呼び出されます。

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。

このタスクについて

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。

手順

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 HANDLER-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。このコンテナーについて詳しくは、 178 ページの『DFHERROR コンテナー』を参照してください。

   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。

タスクの結果

メッセージ・ハンドラーがエラーから回復できる場合は、パイプライン処理は正常に続行されます。 回復できない場合は、CICS は、エラーに関する情報が含まれた SOAP 障害を生成します。 トランザクションが異常終了する場合は、障害に異常終了コードが含められます。

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。

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、ハンドラーが完全な 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、パイプライン構成ファイルで指定されます。 Java をサポートしていないパイプラインの場合、メッセージ・ハンドラー <cics_soap_1.1_handler> または <cics_soap_1.2_handler> を指定する必要があります。Java をサポートしているパイプラインの場合、メッセージ・ハンドラー <cics_soap_1.1_handler_java> または <cics_soap_1.2_handler_java> を指定する必要があります。

通常、1 つのパイプラインに必要な SOAP ハンドラーは 1 つだけです。 ただし、状況によっては、複数の SOAP ハンドラーが必要です。 例えば、複数の SOAP ハンドラーを定義すれば、SOAP ヘッダーを特定の順序で処理できるようになります。

メッセージ・ハンドラー <cics_soap_1.1_handler> と <cics_soap_1.2_handler>、またはメッセージ・ハンドラー <cics_soap_1.1_handler_java> と <cics_soap_1.2_handler_java> を同じパイプラインに定義してはなりません。 パイプラインが SOAP 1.1 と SOAP 1.2 の両方のメッセージを処理することが予想される場合、メッセージ・ハンドラー <cics_soap_1.2_handler> または <cics_soap_1.2_handler_java> を使用してください。

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 および SOAP 1.2 メッセージ・ハンドラー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・プログラムによって検査または変更できる情報 (プログラムが呼び出される SOAP ヘッダー・ブロックや SOAP メッセージ本体など) が保持されます。

チャネルと、ヘッダー処理プログラムが使用できるコンテナーについては、 174 ページの『ヘッダー処理プログラム・インターフェース』で説明します。

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

ヘッダー処理プログラムは、通常トランザクション CPIH の下で実行されます。CPIH は属性 TASKDATALOC(ANY) で定義されます。したがって、プログラムをリンク・エディットする際は、AMODE(31) オプションを指定する必要があります。

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>、<cics_soap_1.2_handler>、<cics_soap_1.1_handler_java>、および <cics_soap_1.2_handler_java> エレメントには、0 または 1 つ以上の <headerprogram> エレメントが含まれており、このエレメントのそれぞれには、以下の子が含まれます。

<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、以下の項目によって決まります。

- <namespace>、<localname>、および <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処理のために選択される SOAP メッセージの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。
  - 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。
  - その他の規則によって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、処理について適格です。

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、処理について適格です。

```
http://www.w3.org/2003/05/soap-envelope/role/next
```

```
http://www.w3.org/2003/05/soap-envelope/role/ultimateReceiver
```

処理について適格なヘッダー・ブロックは、次の規則によって選択されるまで処理されません。

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

前の規則に基づく処理について適格なヘッダー・ブロックは、次の条件が満たされた場合にのみ、処理のために選択されます。

- 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
- 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内の <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" ...> .... </t: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> エレメントに * 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一連のコンテキスト・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、ヘッダー処理プログラム・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は、DFHWS-XMLNS、DFHWS-BODY、および DFHXMLSS-PARSE です。

注**:** Axis2 を使用して SOAP メッセージを処理する Web サービスでは、ヘッダー処理プログラム・インターフェースを使用することもできますが、Java で独自の Axis2 ハンドラーを作成して SOAP ヘッダーを処理する方がより効果的です。 Axis2 ハンドラーの作成については、Writing Your Own Axis2 Module を参照してください。

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。 ヘッダー・ブロックを持たない 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に、このようになります。

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。

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、この情報を使用して以下の作業を実行できます。

- 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。 以下に例を示します。

```
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

このコンテナーは、SOAP エンベロープの本体部分を格納します。 ヘッダー処理プログラムは、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。

コンテナー DFHXMLSS-PARSE

パイプライン構成で `<cics_soap_1.1_handler>` または `<cics_soap_1.2_handler>` いずれかのエレメントを使用する場合、ヘッダー・プログラムが呼び出されると、DFHXMLSS-PARSE にはそのヘッダーの XML システム・サービス (XMLSS) レコードが格納されます。 `<cics_soap_1.1_handler_java>` または `<cics_soap_1.2_handler_java>` エレメントを使用する場合、このコンテナーは作成されません。

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

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。

これらのコンテナーについて詳しくは、 177 ページの『パイプラインで使用されるコンテナー』を参照してください。

端末ハンドラーでのインバウンド要求の動的ルーティング

サービス・プロバイダー・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・ハンドラーの 1 つであるとき、コンテナー **DFHWS-APPHANDLER** に指定されるターゲット・アプリケーションのハンドラー・プログラムが、動的ルーティングに適格である場合があります。アプリケーション・ハンドラー・プログラムより前のパイプライン処理はすべて、常に SOAP メッセージを受け取った CICS 領域でローカルに実行されます。

ターゲット・アプリケーションのハンドラー・プログラムを実行するトランザクションは、以下の条件のいずれかが真である場合に、ルーティングに適格です。

- パイプラインがメッセージを処理するトランザクションが、DYNAMIC または REMOTE として定義される。このトランザクションは、インバウンド SOAP メッセージからの URI のマップに使用される URIMAP に定義されます。
- パイプラインのプログラムが、コンテナー **DFHWS-USERID** の内容を初期値から変更した。
- パイプラインのプログラムが、コンテナー **DFHWS-TRANID** の内容を初期値から変更した。
- WS-AT SOAP ヘッダーがインバウンド SOAP メッセージ内にある。

前のすべてのシナリオで、パイプラインの処理中にタスク切り替えが起こります。2 番目のタスクは、**DFHWS-TRANID** コンテナーに指定されたトランザクションの下で実行します。このタスク切り替えによって動的ルーティングが起こる機会が生じますが、CICS 領域同士の接続に MRO が使用される場合に限ります。 さらに、ルーティング先の CICS 領域が、チャネルおよびコンテナーをサポートする必要があります。

DFHWS-TRANID に指定されたトランザクションの TRANSACTION 定義が以下の一連の属性のいずれかを指定する場合にのみ、ルーティングが起こります。

DYNAMIC(YES)
: トランザクションは、ルーティング・プログラムが **DSRTPGM** システム初期設定パラメーターに指定された、分散ルーティング・モデルを使用してルーティングされます。

DYNAMIC(NO) REMOTESYSTEM(*sysid*)
: トランザクションは、*sysid* で識別されるシステムにルーティングされます。

Web サービス要求のルーティングについて詳しくは、技術情報 Routing of provider mode CICS web services を参照してください。

CICS Web サービス・アシスタントを使用して配置されたアプリケーションでは、CICS がユーザー・プログラムにリンクする際に、要求を動的にルーティングする 2 番目の機会があります。 このとき要求は、ルーティング・プログラムが **DTRPGM** システム初期設定パラメーターに指定された、動的ルーティング・モデルを使用してルーティングされます。 このケースでは、プログラムの特性によってルーティングが適格であると判断されます。プログラムにリンクする際にチャネルおよびコンテナーを使用する場合は、V3.1 以上の CICS 領域にのみ要求を動的にルーティングできます。COMMAREA を使用する場合は、この制限は適用されません。

要求がターゲット領域に動的にルーティングされると、トランザクションが ROUTABLE(YES) および DYNAMIC(YES) として定義されていても、その要求をターゲット領域から第 3 の領域へ動的にルーティングすることはできません。ただし、トランザクションを静的にターゲット領域から第 3 の領域へルーティングすることは可能です。

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。

次のようにコンテナーを分類することができます。

制御コンテナー
: このコンテナーは、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
: このコンテナーは、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。コンテキスト・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。

ヘッダー処理プログラムのコンテナー
: このコンテナーには、CICS 提供の SOAP メッセージ・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。これらのコンテナーについて詳しくは、ヘッダー処理プログラム・インターフェースを参照してください。

セキュリティー・コンテナー
: このコンテナーは、Trust クライアント・インターフェースとセキュリティー・メッセージ・ハンドラーが、Security Token Service (STS) を使用してセキュリティー・トークンを処理するために使用する情報が含まれます。セキュリティー・コンテナーの名前は CICS によって定義されます。名前は DFH という文字で始まります。

生成されたコンテナー
: このコンテナーは、処理のためにアプリケーション・プログラムとの間で受け渡しされる、変数配列や長ストリングなどの SOAP メッセージからのデータが含まれます。 CICSは、パイプライン処理中にこれらのコンテナーを自動的に作成し、名前は DFH という文字で始まります。

ユーザー・コンテナー
: このコンテナーは、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。

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・ハンドラーに伝達する、DATATYPE(BIT) のコンテナーです。

表 *5. DFHERROR* コンテナーの構造

フィールド名	長さ **(**バイト**)**	内容
PIISNEB-MAJOR-VERSION	1	『1』
PIISNEB-MINOR-VERSION	1	『1』
PIISNEB-ERROR-TYPE	1	エラーのタイプを示す数値。値については 179 ページの表 6 で説明します。
PIISNEB-ERROR-MODE	1	**P** プロバイダー・パイプラインで起こったエラー **R** リクエスター・パイプラインで起こったエラー **T** Trust クライアントで起こったエラー

表 *5. DFHERROR* コンテナーの構造 (続き)

フィールド名	長さ (バイト)	内容
PIISNEB-ABCODE	4	エラーがトランザクションの異常終了に関連する場合の異常終了コード。
PIISNEB-ERROR-CONTAINER1	16	エラーがコンテナーに関連する場合のコンテナーの名前。
PIISNEB-ERROR-CONTAINER2	16	エラーが複数のコンテナーに関連する場合の、2 番目のコンテナーの名前。
PIISNEB-ERROR-NODE	8	エラーが発生したハンドラー・プログラムの名前。

表 *6. PIISNEB-ERROR-TYPE* フィールドの値

PIISNEB-ERROR-TYPE の値	意味
1	ハンドラー・プログラムは失敗しました。異常終了コードはフィールド PIISNEB-ABCODE にあります。
2	ハンドラーで必要なコンテナーが空でした。コンテナーの名前はフィールド PIISNEB-ERROR-CONTAINER1 にあります。
3	ハンドラーで必要なコンテナーが欠落していました。コンテナーの名前はフィールド PIISNEB-ERROR-CONTAINER1 にあります。
4	1 つのコンテナーしか予想されていないときに、ハンドラーに 2 つのコンテナーが渡されました。 コンテナーの名前はフィールド PIISNEB-ERROR-CONTAINER1 および PIISNEB-ERROR-CONTAINER2 にあります。
5	ターゲット・プログラムへのリンクに失敗しました。ターゲット・プログラムが失敗した場合、異常終了コードはコンテナー PIISNEB-ABCODE にあります。
6	基礎トランスポートのエラーのために、パイプライン・マネージャーがリモート・サーバーとの通信に失敗しました。
7	DFHWS-STSACTION コンテナーにエラーがあります。欠落または破損しているか、間違った値が含まれています。
8	DFHPIRT はパイプラインの開始に失敗しました。
9	DFHPIRT はコンテナーにメッセージを書き込もうとして失敗しました。

表 6. *PIISNEB-ERROR-TYPE* フィールドの値 (続き)

PIISNEB-ERROR-TYPE の値	意味
10	DFHPIRT はコンテナーからメッセージを取得しようとして失敗しました。
11	未処理エラーが発生しました。

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 PIISNEB-MINOR-VERSION PIC X(1).
  02 PIISNEB-ERROR-TYPE PIC X(1).
  02 PIISNEB-ERROR-MODE 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).
```

以下の表は、コンテナーと対応する言語コピーブックを示しています。

表 7. コンテナーと対応するコピーブック

言語	サンプル集
COBOL	DFHPIUCO
PL/I	DFHPIUCL
C および C++	dfhpiuch.h
アセンブラー	DFHPIUCD

DFHFUNCTION コンテナー:

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する、DATATYPE(CHAR) のコンテナーです。

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ハンドラーで、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。 図 22 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。

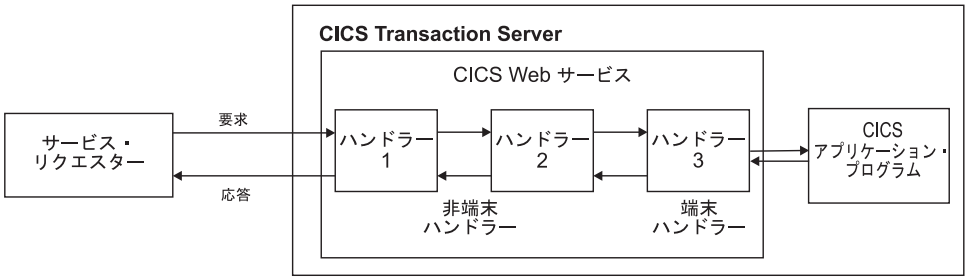


図 22. サービス・プロバイダー・パイプラインでのハンドラーの順序

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

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 RECEIVE-RESPONSE です。 182 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。

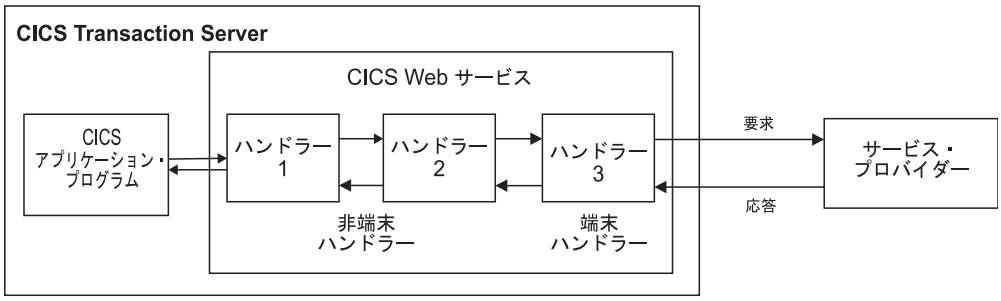


図 23. サービス・リクエスター・パイプラインでのハンドラーの順序

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

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

DFHHTTPMETHOD コンテナー**:**

これは、HTTP プロバイダー・モードのすべての CICS パイプライン内のアプリケーション・プログラムで使用できる DATATYPE(CHAR) のコンテナーです。

長さが 8 文字であるこのコンテナーは、着信要求で使用された HTTP メソッドの名前を保持します。要求が HTTP を介して着信しなかった場合、データは取り込まれません。

DFHHTTPSTATUS コンテナー**:**

DFHHTTPSTATUS は、サービス・プロバイダー・パイプラインの応答段階で生成されるメッセージの HTTP 状況コードと状況テキストを指定するために使われる、DATATYPE(CHAR) のコンテナーです。

DFHHTTPSTATUS コンテナーの内容は、以下のような構造の、HTTP 応答メッセージの初期状況表示行と同じでなければなりません。

```
HTTP/1.1 nnn tttttttt
```

HTTP/1.1
HTTP のバージョンとリリース。

nnn 戻される HTTP 状況コード (3 桁の 10 進数)。

tttttttt
状況コード nnn に関連した、人が読んで理解できる状況テキスト。

内容は、例えば次のストリングのようになります。

```
HTTP/1.1 412 Precondition Failed
```

パイプラインが WebSphere MQ トランスポートを使用する場合、DFHHTTPSTATUS コンテナーは無視されます。

コンテナーに 45 バイトを超えるデータが含まれている場合、CICS は 45 バイトを送信し、残りのデータは無視します。

DFHMEDIATYPE コンテナー**:**

DFHMEDIATYPE は、サービス・プロバイダー・パイプラインの応答段階で生成されるメッセージのメディア・タイプを指定するために使われる、DATATYPE(CHAR) のコンテナーです。

DFHMEDIATYPE コンテナーの内容は、スラッシュ文字で区切られたタイプおよびサブタイプである必要があります。 以下の 2 つのストリングは、DFHMEDIATYPE コンテナーの内容として正しい例を示しています。

```
text/plain
image/svg+xml
```

パイプラインが WebSphere MQ トランスポートを使用する場合、DFHMEDIATYPE コンテナーは無視されます。

DFHNORESPONSE コンテナー**:**

DFHNORESPONSE は、サービス・リクエスター・パイプラインの要求段階で、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す、DATATYPE(CHAR) のコンテナーです。

DFHNORESPONSE コンテナー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。

- コンテナー DFHNORESPONSE が存在する場合は、応答は見込まれません。

- コンテナー DFHNORESPONSE が不在の場合は、応答が見込まれます。

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 (または、存在しない場合に作成すること) はお勧めできません。エンドポイント間のプロトコルを乱す可能性があるためです。

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。

DFHREQUEST コンテナー**:**

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する DATATYPE(CHAR) のコンテナーです。

CICS 提供 SOAP メッセージ・ハンドラーの 1 つがパイプラインで構成されている場合は、コンテナー DFHREQUEST が更新されて SOAP エンベロープに SOAP メッセージ・ヘッダーが組み込まれます。 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 エンベロープが格納され、そのすべての内容が UTF-8 コード・ページに入ります。

DFHREQUEST コンテナーは、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、DFHFUNCTION コンテナーには `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。 パイプライン要求段階の処理が正常に続行され、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) の処理が行われます。

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、DFHRESPONSE コンテナーに応答を書き込むことができます。 このようにして、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。

DFHRESPONSE コンテナー**:**

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する DATATYPE(CHAR) のコンテナーです。 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。

DFHRESPONSE コンテナーは、メッセージ・ハンドラーが呼び出されるときに存在し、DFHFUNCTION コンテナーには `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され、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) の処理が行われます。

DFHFUNCTION コンテナーに RECEIVE-REQUEST、SEND-REQUEST、PROCESS-REQUEST、または HANDLER-ERROR が格納されているとき、DFHRESPONSE コンテナーも存在しますが、長さはゼロです。

DFHWS-CCSID コンテナー**:**

DFHWS-CCSID は、応答コンテナー内のデータの CCSID を指定するフルワード (4 バイト) を含む、DATATYPE(BIT) のコンテナーです。

このコンテナーは、CICS コードを使用して言語構造を XML に変換するプロバイダー・モード・パイプラインでのみ有効です。

この CCSID は、WSBIND ファイルを生成するために使用される CCSID と互換性がなければなりません。 互換性がない場合、生成される SOAP 応答に間違った文字または無効文字が含まれる可能性があります。

CCSID を 930、1390、5026、または 1026 に変更することはできません。または、CCSID が現在それらの値のいずれかである場合、それ以外の値に変更することはできません。 また、CICS では、CCSID をクライアントの CCSID として使用可能な値に変更することはできません。

DFHWS-CCSID コンテナー内の値の処理に何らかの問題が生じると、WSBIND ファイルからの CCSID を使用して処理が継続されます。

DFHWS-CCSID コンテナーは、チャネル主導型アプリケーション・プログラムから戻されたときのみ検査されます。

DFHWS-NODEJSAPP コンテナー**:**

DFHWS-NODEJSAPP は、このパイプラインの NODEJSAPP リソースの名前を格納する DATATYPE(CHAR) のコンテナーです。

DFHWS-NODEJSAPP は、このパイプラインの NODEJSAPP リソースの名前を格納する DATATYPE(CHAR) のコンテナーです。このコンテナーは、プロバイダー・モード・パイプラインの場合にのみ有効で、しかも、パイプラインが NODEJSAPP リソースを介して開始される場合にのみ有効です。

コンテナーがパイプライン・プロトコルを制御する方法

DFHFUNCTION、DFHREQUEST、および DFHRESPONSE コンテナーの内容が一緒にパイプライン・プロトコルを制御します。

パイプラインの実行の 2 つの段階 (要求段階と応答段階) で、DFHFUNCTION の値が、各メッセージ・ハンドラーに渡される制御コンテナーを決定します。

DFHFUNCTION	コンテキスト	**DFHREQUEST**	**DFHRESPONSE**
RECEIVE-REQUEST	サービス・プロバイダー、要求段階	存在 (長さ > 0)	存在 (長さ = 0)

DFHFUNCTION	コンテキスト	DFHREQUEST	DFHRESPONSE
SEND-RESPONSE	サービス・プロバイダー、応答段階	不在	存在 (長さ > 0)
SEND-REQUEST	サービス・リクエスター、要求段階	存在 (長さ > 0)	存在 (長さ = 0)
RECEIVE-RESPONSE	サービス・リクエスター、応答段階	不在	存在 (長さ > 0)
PROCESS-REQUEST	サービス・プロバイダー、端末ハンドラー	存在 (長さ > 0)	存在 (長さ = 0)
HANDLER-ERROR	サービス・リクエスターまたはサービス・プロバイダー、どちらかの段階	不在	存在 (長さ = 0)
NO-RESPONSE	サービス・リクエスターまたはサービス・プロバイダー、応答段階	不在	不在

その後の処理は、メッセージ・ハンドラーがパイプラインに戻すコンテナーによって決定されます。

要求段階中

- メッセージ・ハンドラーは DFHREQUEST コンテナーを戻すことができます。 処理は次のハンドラーを使用して要求段階で続行します。 コンテナー内のデータの長さをゼロにしてはなりません。
- メッセージ・ハンドラーは DFHRESPONSE コンテナーを戻すことができます。 処理は応答段階に切り替わり、DFHFUNCTION をサービス・プロバイダーで SEND-RESPONSE に、サービス・リクエスターで RECEIVE-RESPONSE に設定して、同じハンドラーが呼び出されます。 コンテナー内のデータの長さをゼロにしてはなりません。
- メッセージ・ハンドラーはコンテナーを戻すことができません。 処理は応答段階に切り替わり、DFHFUNCTION を NO-RESPONSE に設定して、同じハンドラーが呼び出されます。

端末ハンドラーで **(**サービス・プロバイダーのみ**)**

- メッセージ・ハンドラーは DFHRESPONSE コンテナーを戻すことができます。 処理は応答段階に切り替わり、DFHFUNCTION の新しい値 (SEND-RESPONSE) で、前のハンドラーが呼び出されます。 コンテナー内のデータの長さをゼロにしてはなりません。
- メッセージ・ハンドラーはコンテナーを戻すことができません。 処理は応答段階に切り替わり、DFHFUNCTION の新しい値 (NO-RESPONSE) で、前のハンドラーが呼び出されます。

応答段階中

- メッセージ・ハンドラーは DFHRESPONSE コンテナーを戻すことができます。 処理は応答段階で続行し、次のハンドラーが呼び出されます。コンテナー内のデータの長さをゼロにしてはなりません。
- メッセージ・ハンドラーはコンテナーを戻すことができません。 処理は応答段階で続行し、DFHFUNCTION の新しい値 (NO-RESPONSE) で、シーケンスの次のハンドラーが呼び出されます。

重要: 要求段階で、メッセージ・ハンドラーは DFHREQUEST または DFHRESPONSE を戻すことができますが、両方を戻すことはできません。メッセージ・ハンドラーが呼び出されるときに両方のコンテナーが存在しているので、どちらかを削除する必要があります。
次の表に、DFHFUNCTION のすべての値と、各メッセージ・ハンドラーによって戻される DFHREQUEST と DFHRESPONSE のすべての組み合わせについて、パイプラインによってとられるアクションを示します。

DFHFUNCTION	コンテキスト	**DFHREQUEST**	**DFHRESPONSE**	アクション
RECEIVE-REQUEST	サービス・プロバイダー、要求段階	存在 (長さ > 0)	存在	(エラー)
RECEIVE-REQUEST	サービス・プロバイダー、要求段階	存在 (長さ > 0)	不在	RECEIVE-REQUEST 関数で次のハンドラーを呼び出す
RECEIVE-REQUEST	サービス・プロバイダー、要求段階	存在 (長さ = 0)	適用外	(エラー)
RECEIVE-REQUEST	サービス・プロバイダー、要求段階	不在	存在 (長さ > 0)	応答段階に切り替え、SEND-RESPONSE 関数で同じハンドラーを呼び出す
RECEIVE-REQUEST	サービス・プロバイダー、要求段階	不在	存在 (長さ = 0)	(エラー)
RECEIVE-REQUEST	サービス・プロバイダー、要求段階	不在	不在	NO-RESPONSE 関数で同じハンドラーを呼び出す
SEND-RESPONSE	サービス・プロバイダー、応答段階	適用外	存在 (長さ > 0)	SEND-RESPONSE 関数で前のハンドラーを呼び出す
SEND-RESPONSE	サービス・プロバイダー、応答段階	適用外	存在 (長さ = 0)	(エラー)
SEND-RESPONSE	サービス・プロバイダー、応答段階	適用外	不在	NO-RESPONSE 関数で同じハンドラーを呼び出す
SEND-REQUEST	サービス・リクエスター、要求段階	存在 (長さ > 0)	存在 (長さ ≥ 0)	(エラー)
SEND-REQUEST	サービス・リクエスター、要求段階	存在 (長さ > 0)	不在	SEND-REQUEST 関数で次のハンドラーを呼び出す
SEND-REQUEST	サービス・リクエスター、要求段階	存在 (長さ = 0)	適用外	(エラー)
SEND-REQUEST	サービス・リクエスター、要求段階	不在	存在 (長さ > 0)	応答段階に切り替え、RECEIVE-RESPONSE 関数で前のハンドラーを呼び出す
SEND-REQUEST	サービス・リクエスター、要求段階	不在	存在 (長さ = 0)	(エラー)
SEND-REQUEST	サービス・リクエスター、要求段階	不在	不在	NO-RESPONSE 関数で同じハンドラーを呼び出す
RECEIVE-RESPONSE	サービス・リクエスター、応答段階	適用外	存在 (長さ > 0)	RECEIVE-RESPONSE 関数で前のハンドラーを呼び出す

DFHFUNCTION	コンテキスト	DFHREQUEST	DFHRESPONSE	アクション
RECEIVE-RESPONSE	サービス・リクエスター、応答段階	適用外	存在 (長さ = 0)	(エラー)
RECEIVE-RESPONSE	サービス・リクエスター、応答段階	適用外	不在	NO-RESPONSE 関数で同じハンドラーを呼び出す
PROCESS-REQUEST	サービス・プロバイダー、端末ハンドラー	適用外	存在 (長さ > 0)	RECEIVE-RESPONSE 関数で前のハンドラーを呼び出す
PROCESS-REQUEST	サービス・プロバイダー、端末ハンドラー	適用外	存在 (長さ = 0)	(エラー)
PROCESS-REQUEST	サービス・プロバイダー、端末ハンドラー	適用外	不在	NO-RESPONSE 関数で同じハンドラーを呼び出す
HANDLER-ERROR	サービス・リクエスターまたはサービス・プロバイダー、どちらかの段階	適用外	存在 (長さ > 0)	SEND-RESPONSE 関数または RECEIVE-RESPONSE 関数で前のハンドラーを呼び出す
HANDLER-ERROR	サービス・リクエスターまたはサービス・プロバイダー、どちらかの段階	適用外	存在 (長さ = 0)	(エラー)
HANDLER-ERROR	サービス・リクエスターまたはサービス・プロバイダー、どちらかの段階	適用外	不在	NO-RESPONSE 関数で同じハンドラーを呼び出す

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処理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。

DFH-EXIT-HEADER1 コンテナー**:**

DFH-EXIT-HEADER1 は DATATYPE(CHAR) のコンテナーです。 ここには、CICS 内の Web サービス・プロバイダー・アプリケーションから応答に追加される、1 つ以上の SOAP ヘッダーが格納されます。

グローバル・ユーザー出口 XWSPRRWO を実行するプログラムは、ヘッダーを SOAP 応答に追加できます。 ヘッダーは有効な SOAP でなければならず、名前空

間はヘッダー XML で自己完結型でなければなりません。 データをこのコンテナーに入れるプログラムは、その有無を検査し、新しいヘッダーをデータの末尾に追加する必要があります。 このベスト・プラクティスに従い、必要に応じて、同じ出口点で複数のプログラムを動かすことができます。

DFH-HANDLERPLIST コンテナー**:**

DFH-HANDLERPLIST は、パイプライン構成ファイルの適切な <handler_parameter_list> エレメントの内容で初期化される DATATYPE(CHAR) のコンテナーです。

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。

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する DATATYPE(CHAR) のコンテナーです。

パイプライン構成ファイルにサービス・パラメーター・リストを指定していない場合、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。

DFHWS-APPHANDLER コンテナー**:**

DFHWS-APPHANDLER は、サービス・プロバイダー・パイプラインで、パイプライン構成ファイルの <apphandler> エレメントの内容で初期化される DATATYPE(CHAR) のコンテナーです。

<apphandler> エレメントを含むパイプラインの端末ハンドラーで、提供されている 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。

関連概念:

113 ページの『アプリケーション・ハンドラー』
アプリケーション・ハンドラーは、実行時に SOAP サービス・プロバイダー・パイプラインの端末ハンドラーがリンクする CICS プログラムです。

DFHWS-APPHANCLAS コンテナー**:**

DFHWS-APPHANCLAS は、サービス・プロバイダー・パイプラインで、パイプライン構成ファイルの <apphandler_class> エレメントの内容で初期化される DATATYPE(CHAR) のコンテナーです。

Java ベース・パイプラインの端末ハンドラーで、提供されている SOAP ハンドラー、<cics_soap_1.1_handler_java> および <cics_soap_1.2_handler_java> 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。

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。

関連概念:

113 ページの『アプリケーション・ハンドラー』
アプリケーション・ハンドラーは、実行時に SOAP サービス・プロバイダー・パイプラインの端末ハンドラーがリンクする CICS プログラムです。

関連資料:

114 ページの『<apphandler_class> パイプライン構成エレメント』
パイプラインの端末ハンドラーが Axis2 アプリケーション・ハンドラーにリンクすることを指定します。

DFHWS-DATA コンテナー**:**

DFHWS-DATA は、CICS Web サービス・アシスタントで配置されるサービス・リクエスター・アプリケーション (およびオプションでサービス・プロバイダー・アプリケーション) で使用される、DATATYPE(BIT) のコンテナーです。このコンテナーは、SOAP 要求と相互にマップする最上位のデータ構造を格納します。

サービス・リクエスター・アプリケーションでは、サービス・リクエスター・プログラムが **EXEC CICS INVOKE SERVICE** コマンドを出すときに、DFHWS-DATA コンテナーが存在している必要があります。 このコマンドが出されると、CICS は、コンテナーにあるデータ構造を SOAP 要求に変換します。 SOAP 応答が受信されると、 CICS はこれを、同じコンテナーにあるアプリケーションに戻される別のデータ構造に変換します。

サービス・プロバイダー・アプリケーションでは、DFHLS2WS または DFHWS2LS バッチ・ジョブで **CONTID** パラメーターを指定しないと、DFHWS-DATA コンテナーがデフォルトで使用されます。 CICS は、SOAP 要求メッセージを、DFHWS-DATA コンテナー内にあるアプリケーションに渡されるデータ構造に変換します。 次に応答が同じコンテナーに保管され、CICS がデータ構造を SOAP 応答メッセージに変換します。

DFHWS-FAULT コンテナー**:**

DFHWS-FAULT は、CICS が生成する SOAP 障害のタイプに関する情報を保持する DATATYPE(BIT) のコンテナーです。

このコンテナーに保持されるバイナリー・フルワードは、その後の Web サービス応答の処理に使用できる障害タイプを以下のように示します。

1. 最新の SOAP 障害は CICS 障害に関するものでした (例えば CICS またはユーザーの異常終了)。

2. 最新の SOAP 障害はアプリケーション障害に関するものでした。 EXEC CICS SOAPFAULT DELETE コマンドの発行時にコンテナーが削除されます。 2 番目の (または新しい) SOAP 障害が作成される場合、CICS は新しいコンテナーを適切に更新します。

このコンテナーの内容を変更することはできません。

DFHWS-LOCATION コンテナー**:**

DFHWS-LOCATION は、HTTP 応答が 302、303、または 307 だったときに提供されたロケーション・ヘッダーを格納する DATATYPE(CHAR) のコンテナーです。

DFHWS-MEP コンテナー**:**

DFHWS-MEP は、インバウンドまたはアウトバウンドの SOAP メッセージのメッセージ交換パターン (MEP) についての代表値を格納する、DATATYPE(BIT) のコンテナーです。 この値の長さは 1 バイトです。

CICS は、サービス・リクエスターとサービス・プロバイダーの両方について、 4 つのメッセージ交換パターンをサポートします。 メッセージ交換パターンは Web サービスについて WSDL 2.0 文書で定義され、CICS がプロバイダーとして応答するべきかどうか、および CICS が外部プロバイダーからの応答を期待するべきかどうかを決定します。 リクエスター・モードでは、CICS が応答を待機する時間は PIPELINE リソースを使用して構成されます。

CICS Web サービス・アシスタントを使ってアプリケーションを配置した場合、CICS によってこのコンテナーのデータが設定されます。

- サービス・プロバイダー・パイプラインでは、このコンテナーは、端末ハンドラーからインバウンド・メッセージを受け取るときに、DFHPITP アプリケーション・ハンドラーによってデータを設定されます。
- サービス・リクエスター・パイプラインでは、このコンテナーは、アプリケーションが **INVOKE SERVICE** コマンドを使用するときにデータを設定されます。

アプリケーションが DFHPIRT チャネルを使用してパイプラインを開始する場合、アプリケーションがこのコンテナーのデータを設定します。 コンテナーが存在しないかコンテナーに値がない場合、チャネルに DFHNORESPONSE コンテナーが存在するかどうかに応じて、CICS は要求が In-Out または In-Only MEP のいずれかを使用していると想定します。

このコンテナーには、提供されているアプリケーション・ハンドラー・プログラム DFHPITP によってデータが入れられます。 別のアプリケーション・ハンドラーを使用する場合、このコンテナーを使用することはできません。

表 *8.* コンテナー *DFHWS-MEP* に表示される値

値	**MEP**	**URI**
1	In-Only	http://www.w3.org/ns/wsdl/in-only
2	In-Out	http://www.w3.org/ns/wsdl/in-out
4	Robust-In-Only	http://www.w3.org/ns/wsdl/robust-in-only
8	In-Optional-Out	http://www.w3.org/ns/wsdl/in-opt-out

DFHWS-OPERATION コンテナー**:**

DFHWS-OPERATION は、CICS Web サービス・アシスタントで配置されるサービス・プロバイダー・アプリケーションで通常使用される、DATATYPE(CHAR) のコンテナーです。 SOAP 要求で指定される操作の名前を格納します。

サービス・プロバイダーでは、アプリケーションが呼び出される操作の名前をこのコンテナーが指定します。 提供されている SOAP メッセージ・ハンドラーがターゲット・アプリケーション・プログラムに制御を渡すときに、このコンテナーにデータが設定され、ターゲット・プログラムがチャネル・インターフェースを使用して呼び出されるときのみ、このコンテナーが表示されます。

サービス・リクエスター・パイプラインでは、このコンテナーは、**EXEC CICS INVOKE SERVICE** コマンドの OPERATION オプションに指定される名前を格納します。 このコンテナーは、このコマンドを出すアプリケーションでは使用できません。

このコンテナーには、提供されているアプリケーション・ハンドラー・プログラム DFHPITP によってデータが入れられます。 別のアプリケーション・ハンドラーを使用する場合、このコンテナーを使用することはできません。

DFHWS-PIPELINE コンテナー**:**

DFHWS-PIPELINE は、プログラムが実行されている PIPELINE の名前を格納する、 DATATYPE(CHAR) のコンテナーです。

このコンテナーの内容を変更することはできません。

DFHWS-RESPWAIT コンテナー**:**

DFHWS-RESPWAIT は DATATYPE(BIT) のコンテナーであり、アウトバウンド Web サービス要求メッセージおよび応答メッセージに適用されるタイムアウト (秒) を表す符号なしフルワード 2 進数がこれに入ります。

このコンテナーの値は、PIPELINE 定義の RESPWAIT 属性によって提供され、INVOKE SERVICE コマンドが実行されるとCICS によって設定されます。 INVOKE SERVICE コマンドが実行される前にユーザー・アプリケーションによってこのコンテナーに設定されたすべての値は、無視されます。

パイプライン処理中に呼び出されるメッセージ・ハンドラー・プログラムは DFHWS-RESPWAIT コンテナーの値を上書きすることができます。これが行われると、更新された値は、DEFT またはブランクに設定されていない RESPWAIT 属性が PIPELINE 定義に含まれる場合にのみ使用されます。 DEFT またはブランクに設定された RESPWAIT 属性が PIPELINE 定義に含まれる場合には、DFHWS-RESPWAIT コンテナー内の値に関係なく、トランスポート・プロトコルのタイムアウト値が常に使用されます。

このコンテナーはリクエスター・モードのパイプラインでのみ使用されます。

DFHWS-SOAPLEVEL コンテナー**:**

DFHWS-SOAPLEVEL は、処理するメッセージで使用される SOAP のレベルについての情報を格納する、DATATYPE(BIT) のコンテナーです。

このコンテナーには、Web サービス要求または応答で使用される SOAP のレベルを示す、バイナリー・フルワードが格納されます。

1 要求または応答は SOAP 1.1 メッセージです。

2 要求または応答は SOAP 1.2 メッセージです。

10 要求または応答は SOAP メッセージではありません。

このコンテナーの内容を変更することはできません。

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る、DATATYPE(CHAR) のコンテナーです。

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。

同じ JVM サーバーでパイプラインの端末ハンドラーとアプリケーション・ハンドラーの両方を実行する場合、新しいタスクを開始することはできません。 このため、JAX-WS Axis2 アプリケーションを CICS に配置する場合、DFHWS-TRANID を使用してユーザー ID を変更することはできません。

DFHWS-URI コンテナー**:**

DFHWS-URI は、サービスの URI を格納する、DATATYPE(CHAR) のコンテナーです。

サービス・プロバイダー・パイプラインでは、CICS が着信メッセージから相対 URI を抽出し、これを DFHWS-URI コンテナーに入れます。

例えば、Web サービスの URI が `http://example.com/location/address` または `jms://queue?destination=INPUT.QUEUE&targetService=/location/address` の場合、相対 URI は `/location/address` です。

リクエスター・パイプラインで Web サービス・アドレッシングを使用する場合、このコンテナーは以下の順序で作成および更新されます。

1. INVOKE SERVICE コマンドが実行されると、DFHWS-URI コンテナーが作成され、WSDL サービス・エンドポイント・アドレスの値を使ってそれが開始されます。アドレッシング・コンテキストを作成するために WSACONTEXT BUILD API コマンドが使用された場合は、INVOKE SERVICE コマンドでパラメーター URI や URIMAP を指定してはなりません。
2. Web サービス・アドレッシング・ハンドラー (DFHWSADH) の実行時に、非匿名 URI を伴う <wsa:To> EPR がアドレッシング・コンテキスト内に存在す

る場合は、その <wsa:To> EPR の値を使って DFHWS-URI コンテナー内の URI が上書きされます。匿名 URI は無視されます。

SOAP メッセージが、DFHWS-URI の URI によって定義されたサービスに送られます。

サービス・リクエスター・パイプラインでは、CICS は DFHWS-URI コンテナーに、**INVOKE SERVICE** コマンドで指定されている URI を入れます。これが存在しない場合には Web サービス・バインディングの URI を入れます。パイプラインでメッセージ・ハンドラーを使用することにより、この URI を指定変更することができます。

サービスは外部サービスとして HTTP、HTTPS、JMS、または WebSphere MQ URI を使用できます。また、サービスでは、次に示すように別の CICS アプリケーションによって提供されるサービス用の CICS URI を使用することもできます。

URI	照会ストリング	説明
cics://PROGRAM/*program*	?*options*	指定されたプログラムにリンクするために CICS トランスポート・ハンドラーは **EXEC CICS LINK PROGRAM** コマンドを使用し、現在のチャネルとコンテナーを渡します。アプリケーション・データに対するデータ変換は行われません。
cics://SERVICE/*service*	?targetServiceUri=*targetServiceUri* &*options*	プロバイダー・パイプラインを介して要求を実行するために、CICS トランスポート・ハンドラーは (*targetServiceUri* と表される) サービスのパスを使って URIMAP リソースをマッチングします。 この URI タイプを使用する場合には、**targetServiceUri** パラメーターの値を指定する必要があります。
cics://PIPELINE/*pipeline*	?targetServiceUri=*targetServiceUri*	CICS トランスポート・ハンドラーは別のサービス・リクエスター・パイプラインを開始します。

parameter=value という形式 (各パラメーターをアンパーサンドで区切る) を使用して、それぞれのタイプの CICS URI にパラメーターを追加することができます。CICS URI には次のような規則が適用されます。

- 照会ストリング内の最初のパラメーターには、接頭部として疑問符 (?) が必要です。 URI の中で、この場所より前に疑問符 (?) を使うことはできません。
- パラメーター値の中にアンパーサンドを含めるには、文字をエスケープする必要があります。 詳しくは、このトピックの最後に記載されている使用例セクションを参照してください。
- *program* および *pipeline* に小文字の値が含まれている場合、CICS はそれらを大文字に変更します。

リクエスター・パイプラインの終わりで CICS がどのように要求を処理するかは、照会ストリングのパラメーターによって次のように決定されます。

maxCommareaLength=*value*
ターゲット・アプリケーション・プログラムで必要な COMMAREA の最大サイズ (バイト) を指定します。 この値は 32 763 を超えることができません。このパラメーターが照会ストリングに存在する場合、CICS は COMMAREA を使用して指定されたプログラムにリンクします。 このパラメーターが照会ストリングに存在しない場合、CICS はチャネルを使用して指定されたプログラムにリンクします。

このパラメーターでは大/小文字の区別がなく、cics://PROGRAM URI でのみ有効です。

newTask=*yes*|no
トランスポート・ハンドラーが新しいタスクとして要求を実行するかどうかを指定します。

このパラメーターは、大/小文字が区別されません。 `cics://PROGRAM/testapp?newTask=yes` と `cics://PROGRAM/testapp?NEWTASK=Yes` は同じです。

targetServiceUri=*uri*
呼び出されるサービスのパスを指定します。 SERVICE 宛先タイプでは、トランスポート・ハンドラーはサービス・プロバイダー・パイプラインを開始するために `host=localhost` という値を使って URIMAP リソースを見つけます。PIPELINE 宛先タイプでは、トランスポート・ハンドラーは別のリクエスター・パイプラインを開始するためにこの値を使用します。

このパラメーターは、大/小文字が区別されます。

transid=*char(4)*
どのトランザクションで要求が実行されるかを指定します。 トランスポート・ハンドラーは、指定されたトランザクション ID を使って要求ストリームを開始します。

このパラメーターは、大/小文字が区別されます。

userid=*char(8)*
どのユーザー ID で要求が実行されるかを指定します。 トランスポート・ハンドラーは、指定されたユーザー ID を使って要求ストリームを開始します。

このパラメーターは、大/小文字が区別されません。

宛先タイプ	**URI** におけるパラメーター	
PROGRAM	**userid**	オプション
PROGRAM	**transid**	オプション
PROGRAM	**maxCommareaLength**	オプション
PROGRAM	**newTask**	オプション。 **userid** または **transid** を指定する場合、これを yes にするか、または指定しないでください。
PROGRAM	**targetServiceUri**	サポートされていない
SERVICE	**userid**	オプション
SERVICE	**transid**	オプション

宛先タイプ	**URI** におけるパラメーター	
SERVICE	**maxCommareaLength**	サポートされていない
SERVICE	**newTask**	オプション。 **userid** または **transid** を指定する場合、これを yes にするか、または指定しないでください。
SERVICE	**targetServiceUri**	Required
PIPELINE	**userid**	サポートされていない
PIPELINE	**transid**	サポートされていない
PIPELINE	**maxCommareaLength**	サポートされていない
PIPELINE	**newTask**	サポートされていない
PIPELINE	**targetServiceUri**	Required

CICS URI の例

この最初の例では、パイプラインの終わりに達するまでに、以下のような URI が DFHWS-URI コンテナーに入ります。

```
cics://PROGRAM/testapp?newTask=yes&userid=user1
```

トランスポート・ハンドラーは testapp という CICS プログラムにリンクして、チャネルとコンテナーを渡します。データ変換は行われないため、ターゲット・プログラムは現在のチャネル上のコンテナーの内容を処理できる必要があります。 CICS は新しい作業単位、別のユーザー ID user1 のもとでプログラムにリンクします。

この 2 番目の例では、パイプラインの終わりに達するまでに、以下のような URI が DFHWS-URI コンテナーに入ります。

```
cics://SERVICE/getStockQuote?targetServiceUri=/stock/getQuote&newTask=yes&userid=user2
```

トランスポート・ハンドラーは値 `/stock/getQuote` を使って DFHWS-URI コンテナー内の URI を置換し、URI を解決するために **targetServiceUri** パラメーター内のパスを使って URIMAP を検出し、新しいタスクおよび異なるユーザー ID のもとでプロバイダー・パイプラインを開始します。

この 3 番目の例では、パイプラインの終わりに達するまでに、以下のような URI が DFHWS-URI コンテナーに入ります。

```
cics://PIPELINE/reqpipeA?targetServiceUri=cics://PROGRAM/testapp?newTask=yes%26userid=user1
```

トランスポート・ハンドラーは値 `cics://PROGRAM/testapp?newTask=yes &userid=user1` を使って DFHWS-URI コンテナー内の URI を置換し、reqpipeA というリクエスター・パイプラインを開始して現在のチャネルとコンテナーを渡します。文字 `%26` はアンパーサンドをエスケープするため、トランスポート・ハンドラーは URI 全体を DFHWS-URI コンテナーに入れます。

DFHWS-URI-RESID コンテナー**:**

これは、JSON パイプラインによって接続されたアプリケーションでのみ使用可能な、DATATYPE(CHAR) のコンテナーです。

このコンテナーには、URIMAP マッチングで使われたパス URI フラグメントが除去された、URI パス (リソース ID) の簡略コピーが保持されます。以下に例を示します。

着信要求をマッチングした URIMAP に次のような PATH が含まれるとします。

```
/JSONServices/CustomerDetails/*
```

以下の URI がクライアントから着信した場合、

```
http://www.example.org:10000/JSONServices/CustomerDetails/customerNumber/13388?action=query
```

DFHWS-URI-RESID の内容は次のようになります。

```
customerNumber/13388
```

このコンテナーを使用すると、RESTful JSON アプリケーションは、ワイルドカード付き URIMAP を使ってマッチングされる RESTful リソースを示すリソース ID (または 1 次キー) を識別できます。これは、DFHWS-URI の内容を解析するのに比べてかなり簡単です。

注: マッチングを行う URIMAP の PATH 属性にワイルドカードが付いていない場合 (つまり URI を示す完全なパスが含まれていた場合)、このコンテナーの内容は空になります。

注: マッチングを行う URIMAP の PATH 属性には、オプションの照会ストリング・フラグメントが含まれることがあります。その場合、このコンテナーの構築時に照会ストリング・フラグメントは無視されます。

DFHWS-URI-QUERY コンテナー:

これは、HTTP プロバイダー・モードのすべての CICS パイプライン内のアプリケーション・プログラムで使用できる DATATYPE(CHAR) のコンテナーです。

このコンテナーには、URI の照会ストリング・フラグメントが保持されます。以下に例を示します。

以下の URI がクライアントから着信した場合、

```
http://www.example.org:10000/JSONServices/CustomerDetails/customerNumber/13388?action=query&page=1
```

DFHWS-URI-QUERY の内容は次のようになります。

```
action=query&page=1
```

アプリケーションはこのコンテナーの内容を解析して、URI から個々の **name=value** パラメーターを識別することができます。

注: 着信した URI に照会ストリングが含まれない場合、チャネルにはこのコンテナーが存在しません。

DFHWS-URIMAPPATH コンテナー:

これは DATATYPE(CHAR) のコンテナーです。これには、着信 URI のマッチングに使われた URIMAP からの PATH データのコピーが保持されます。

パイプライン接続されたアプリケーションはこのコンテナーを使用して、自身がどのように接続されるようになったかを詳しく認識できます。

DFHWS-USERID コンテナー**:**

DFHWS-USERID は、パイプラインが稼働中のタスクのユーザー ID を使用して初期化される、DATATYPE(CHAR) のコンテナーです。

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-TRANID の内容を変更しない限り、新しいタスクのトランザクション ID は、パイプラインが実行されているタスクと同じになります。

同じ JVM サーバーでパイプラインの端末ハンドラーとアプリケーション・ハンドラーの両方を実行する場合、新しいタスクを開始することはできません。 このため、JAX-WS Axis2 アプリケーションを CICS に配置する場合、DFHWS-USERID を使用してユーザー ID を変更することはできません。

DFHWS-WEBSERVICE コンテナー**:**

DFHWS-WEBSERVICE は、サービス・プロバイダー・パイプラインでのみ使用される、 DATATYPE(CHAR) のコンテナーです。 これは、ターゲット・アプリケーションが 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、実行環境を指定する Web サービスの名前を格納します。

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。

DFHWS-CID-DOMAIN コンテナー**:**

DFHWS-CID-DOMAIN は DATATYPE(CHAR) のコンテナーです。 バイナリー添付ファイルを参照するための content-ID 値の生成に使用されるドメイン・ネームを格納します。

ドメイン名の値はデフォルトでは `cicsts` です。 パイプライン構成ファイルに `<mime_options>` エレメントを指定して、この値を指定変更できます。

このコンテナーの内容を変更することはできません。

DFHWS-MTOM-IN コンテナー**:**

DFHWS-MTOM-IN は、パイプライン構成ファイルの `<cics_mtom_handler>` エレメントに指定されたオプションについての情報と、パイプラインに受信されたメッセージ・フォーマットについての情報を格納する、DATATYPE(BIT) のコンテナーです。

このコンテナーは、パイプラインのインバウンド MTOM メッセージを処理するための情報を格納します。 インバウンド・メッセージは、Web サービス・リクエスターからの要求メッセージか、Web サービス・プロバイダーからの応答メッセージである可能性があります。

パイプライン構成ファイルに <cics_mtom_handler> エレメントを指定しない場合や、MTOM メッセージの代わりに SOAP メッセージが受信される場合は、このコンテナーは作成されません。

Web サービス・セキュリティーがパイプラインで構成されるか、Web サービスについて検証のスイッチが入ると、このコンテナーが作成されるときに、DFHWS-MTOM-IN の XOP_MODE フィールドの内容が CICS によって指定変更されることがあります。例えば、MTOM メッセージの内容を直接モードで処理するためにパイプラインを構成してから、Web サービスについて検証のスイッチを入れると、CICS はパイプライン構成ファイル内に定義された値を指定変更して、互換モードで実行するように XOP 処理を設定します。 CICS が指定変更する理由は、パイプライン内での XOP 文書とバイナリー添付ファイルの処理サポートにおける制限のためです。

このコンテナーの内容を変更することはできません。

表 *9. DFHWS-MTOM-IN* コンテナーの構造

フィールド名	長さ (バイト)	内容
MTOM_STATUS	4	CICS が受信したメッセージが MTOM フォーマットであることを示す値「1」が含まれます。
MTOMNOXOP_STATUS	4	以下のいずれかの値が含まれます。 **0** MTOM メッセージにバイナリー添付ファイルが含まれます。 **1** MTOM メッセージにバイナリー添付ファイルが含まれません。
XOP_MODE	4	以下のいずれかの値が含まれます。 **0** XOP 処理は行われません。 **1** XOP 処理は互換モードで行われます。 **2** XOP 処理は直接モードで行われます。

DFHWS-MTOM-OUT コンテナー:

DFHWS-MTOM-OUT は、パイプライン構成ファイルの <cics_mtom_handler> エレメントに指定されたオプションについての情報を格納する、DATATYPE(BIT) のコンテナーです。

このコンテナーは、パイプライン内のアウトバウンド MTOM メッセージが Web サービス・リクエスターについての応答メッセージであるか Web サービス・プロバイダーについての要求メッセージであるかにかかわらず、これを処理するための情報を格納します。

パイプライン構成ファイルに <cics_mtom_handler> エレメントを指定しない場合や、パイプライン構成ファイルの <mtom_options> エレメントに属性 send_mtom="no" がある場合は、このコンテナーは作成されません。

プロバイダー・モードでは、このコンテナーは DFHWS-MTOM-IN コンテナーと同時に作成されます。 パイプライン構成ファイルの <mtom_options> エレメントに属性 send_mtom="same" がある場合は、 Web サービス・リクエスターにとって MTOM と SOAP どちらの応答メッセージが望ましいのかを示す MTOM_STATUS フィールドが設定されます。

Web サービス・セキュリティーがパイプラインで構成されるか、Web サービスについて検証のスイッチが入ると、このコンテナーが作成されるときに、DFHWS-MTOM-OUT の XOP_MODE フィールドが CICS によって変更されることがあります。 例えば、XOP 文書と任意のバイナリー添付ファイルを直接モードで処理するためにパイプラインを構成してから、Web サービスについて検証のスイッチを入れると、CICS はパイプライン構成ファイル内に定義された値を指定変更して、コンテナーを作成するときに互換モードで実行するように XOP 処理を設定します。 CICS が指定変更する理由は、パイプライン内での XOP 文書とバイナリー添付ファイルの処理サポートにおける制限のためです。

このコンテナーの内容を変更することはできません。

表 *10. DFHWS-MTOM-OUT* コンテナーの構造

フィールド名	長さ (バイト)	内容
MTOM_STATUS	4	MTOM が使用可能かどうかを示します。 **0** MTOM は使用可能ではありません。 アウトバウンド・メッセージは SOAP フォーマットで送信されます。 **1** MTOM は使用可能です。 アウトバウンド・メッセージは MTOM フォーマットで送信されます。
MTOMNOXOP_STATUS	4	バイナリー添付ファイルがない場合に MTOM を使用するかどうかを示します。 **0** バイナリー添付ファイルがない場合に MTOM メッセージを送信しません。 **1** バイナリー添付ファイルがない場合に MTOM メッセージを送信します。
XOP_MODE	4	行われる XOP 処理を示します。 **0** XOP 処理は行われません。 **1** XOP 処理は互換モードで行われます。 **2** XOP 処理は直接モードで行われます。

DFHWS-WSDL-CTX コンテナー**:**

DFHWS-WSDL-CTX は、CICS Web サービス・アシスタントと共に配置されるサービス・プロバイダーまたはサービス・リクエスター・アプリケーションで使用される、DATATYPE(CHAR) のコンテナーです。 これには、モニターに使用できる WSDL コンテキスト情報が保持されます。

DFHWS-WSDL-CTX は、WSDL 文書に関する次のようなコンテキスト情報を保持します。

- アプリケーションが呼び出される操作の名前と名前空間。
- 既知の場合、使用される WSDL 1.1 ポートまたは WSDL 2.0 エンドポイントの名前と名前空間。

これらの値はスペース文字で区切られます。 DFHWS-WSDL-CTX は、ランタイム・レベル 2.1 以降の CICS によってのみデータが設定されます。

CICS Web サービス・アシスタントを使ってアプリケーションを配置した場合、CICS によってこのコンテナーのデータが設定されます。

- サービス・プロバイダー・パイプラインでは、このコンテナーは、端末ハンドラーからインバウンド・メッセージを受け取るときに、DFHPITP アプリケーション・ハンドラーによってデータを設定されます。
- サービス・リクエスター・パイプラインでは、このコンテナーは、アプリケーションが **INVOKE SERVICE** コマンドを使用するときにデータを設定されます。

アプリケーションが DFHPIRT プログラムを使ってパイプラインを開始する場合、アプリケーションは必要に応じて DFHWS-WSDL-CTX コンテナーのデータを設定します。

DFHWS-XOP-IN コンテナー**:**

DFHWS-XOP-IN は DATATYPE(BIT) のコンテナーです。 インバウンド MIME メッセージからアンパックされ、XOP 処理を使用してコンテナーに配置された、バイナリー添付ファイルに対する参照のリストを格納します。

DFHWS-XOP-IN コンテナーの各添付レコードは、次のような項目から成ります。

- バイナリー添付ファイルに関する MIME ヘッダーを格納するコンテナーの 16 バイトの名前
- バイナリー添付ファイルを格納するコンテナーの 16 バイトの名前
- 符号付きハーフワード・バイナリー・フォーマットの、2 バイト長の content-ID
- ASCII 文字ストリングとして保管された、< および > 区切り文字を含む content-ID

このコンテナーの内容を変更することはできません。

DFHWS-XOP-OUT コンテナー**:**

DFHWS-XOP-OUT は DATATYPE(BIT) のコンテナーです。 バイナリー添付ファイルを格納するコンテナーに対する参照のリストが含まれます。バイナリー添付ファイルは、MTOM ハンドラー・プログラムによって、アウトバウンド MIME メッセージにパックされます。

DFHWS-XOP-OUT コンテナーの各添付レコードは、次のような項目から成ります。

- バイナリー添付ファイルに関する MIME ヘッダーを格納するコンテナーの 16 バイトの名前
- バイナリー添付ファイルを格納するコンテナーの 16 バイトの名前

- 符号付きハーフワード・バイナリー・フォーマットの、2 バイト長の content-ID
- ASCII 文字ストリングとして保管された、< および > 区切り文字を含む content-ID

このコンテナーの内容を変更することはできません。

ヘッダー処理プログラムのコンテナー

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一連のコンテキスト・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、ヘッダー処理プログラム・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は、DFHWS-XMLNS、DFHWS-BODY、および DFHXMLSS-PARSE です。

注**:** Axis2 を使用して SOAP メッセージを処理する Web サービスでは、ヘッダー処理プログラム・インターフェースを使用することもできますが、Java で独自の Axis2 ハンドラーを作成して SOAP ヘッダーを処理する方がより効果的です。 Axis2 ハンドラーの作成については、Writing Your Own Axis2 Module を参照してください。

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。 ヘッダー・ブロックを持たない 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に、このようになります。

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。

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、この情報を使用して以下の作業を実行できます。

- 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。 以下に例を示します。

```
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

このコンテナーは、SOAP エンベロープの本体部分を格納します。 ヘッダー処理プログラムは、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。

コンテナー DFHXMLSS-PARSE

パイプライン構成で `<cics_soap_1.1_handler>` または `<cics_soap_1.2_handler>` いずれかのエレメントを使用する場合、ヘッダー・プログラムが呼び出されると、DFHXMLSS-PARSE にはそのヘッダーの XML システム・サービス (XMLSS) レコードが格納されます。 `<cics_soap_1.1_handler_java>` または

<cics_soap_1.2_handler_java> エレメントを使用する場合、このコンテナーは作成されません。

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

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。

これらのコンテナーについて詳しくは、 177 ページの『パイプラインで使用されるコンテナー』を参照してください。

セキュリティー・コンテナー

セキュリティー・コンテナーは、Tivoli Federated Identity Manager などの Security Token Service (STS) との間で ID トークンを送受信するために、DFHWSTC-V1 チャネル上で使用されます。 このインターフェースは *Trust* クライアント・インターフェース と呼ばれ、Web サービス・リクエスターおよびプロバイダーのパイプラインで使用できます。

DFHWS-IDTOKEN コンテナー**:**

DFHWS-IDTOKEN は DATATYPE(CHAR) のコンテナーです。 これには、メッセージの ID トークンを発行するために Security Token Service (STS) によって検証または使用されるトークンが入ります。

トークンは XML 形式でなくてはなりません。

このコンテナーは、Trust クライアント・インターフェースのチャネル DFHWSTC-V1 でのみ使用してください。

DFHWS-RESTOKEN コンテナー**:**

DFHWS-RESTOKEN は DATATYPE(CHAR) のコンテナーです。 Security Token Service (STS) からの応答を格納します。

応答は、DFHWS-STSACTION コンテナーで STS から要求されたアクションに応じて異なります。

- アクションが発行である場合、このコンテナーは、DFHWS-IDTOKEN コンテナーに送信されたものに対して STS が交換したトークンを格納します。
- アクションが検証である場合、このコンテナーは、DFHWS-IDTOKEN コンテナーに送信されたセキュリティー・トークンが有効か無効かを示す URI を保持します。 戻される可能性のある URI は次のとおりです。

URI	説明
`http://schemas.xmlsoap.org/ws/2005/02/trust/status/valid`	セキュリティー・トークンが有効です。
`http://schemas.xmlsoap.org/ws/2005/02/trust/status/invalid`	セキュリティー・トークンが無効です。

このコンテナーは、Trust クライアント・インターフェースを使用する際に、チャネル DFHWSTC-V1 に戻されます。

DFHWS-SERVICEURI コンテナー**:**

DFHWS-SERVICEURI は DATATYPE(CHAR) のコンテナーです。 Security Token Service (STS) が AppliesTo の有効範囲として使用する URI を格納します。

AppliesTo スコープは、セキュリティー・トークンに関連付ける Web サービスを決定するために使用されます。

このコンテナーは、Trust クライアント・インターフェースのチャネル DFHWSTC-V1 でのみ使用してください。

DFHWS-STSACTION コンテナー**:**

DFHWS-STSACTION は DATATYPE(CHAR) のコンテナーです。 セキュリティー・トークンを検証または発行するために、Security Token Service (STS) がとる必要があるアクションの URI を格納します。

このコンテナーで指定できる URI 値は次のとおりです。

URI	説明
`http://schemas.xmlsoap.org/ws/2005/02/trust/Issue`	STS は、DFHWS-IDTOKEN コンテナーに送信されるものと交換にトークンを発行します。
`http://schemas.xmlsoap.org/ws/2005/02/trust/Validate`	STS は、DFHWS-IDTOKEN コンテナーに送信されるトークンを検証します。

このコンテナーは、Trust クライアント・インターフェースのチャネル DFHWSTC-V1 でのみ使用してください。

DFHWS-STSFAULT コンテナー**:**

DFHWS-STSFAULT は DATATYPE(CHAR) のコンテナーです。 Security Token Service (STS) によって戻されたエラーを格納します。

エラーが発生すると、STS が SOAP 障害を出します。 SOAP 障害の内容がこのコンテナーに戻されます。

このコンテナーは、Trust クライアント・インターフェースを使用する際に、チャネル DFHWSTC-V1 に戻されます。

DFHWS-STSREASON コンテナー**:**

DFHWS-STSREASON は DATATYPE(CHAR) のコンテナーです。 このエレメントが Security Token Service (STS) からの応答メッセージに存在する場合は、`<wst:Reason>` エレメントの内容を格納します。

`<wst:Reason>` エレメントには、CICS によって STS に送信された検証要求の状況に関連する情報を提供する、オプションのストリングが含まれます。 セキュリティ

ー・トークンが無効な場合、STS によって提供されたこのエレメント内の情報を調べると、トークンが無効である理由を判別できる可能性があります。

詳しくは、OASIS WS-Trust v1.4 Standard に公開されている「*Web Services Trust Language*」仕様を参照してください。

DFHWS-STSURI コンテナー**:**

DFHWS-STSURI は DATATYPE(CHAR) のコンテナーです。 SOAP メッセージについて ID トークンを検証または発行するために使用する、Security Token Service (STS) の絶対 URI を格納します。

URI の形式は `http://www.example.com:8080/TrustServer/SecurityTokenService` です。セキュリティー要件に応じて、HTTP または HTTPS を使用できます。

このコンテナーは、Trust クライアント・インターフェースのチャネル DFHWSTC-V1 でのみ使用してください。

DFHWS-TOKENTYPE コンテナー**:**

DFHWS-TOKENTYPE は DATATYPE(CHAR) のコンテナーです。 Security Token Service (STS) が SOAP メッセージについて ID トークンとして発行する、要求されたトークン・タイプの URI を格納します。

STS でサポートされている限り、有効などのトークン・タイプでも指定できます。

このコンテナーは、Trust クライアント・インターフェースのチャネル DFHWSTC-V1 でのみ使用してください。

SAML サポート・コンテナー

CICS SAML サポートによって使用される読み取り専用コンテナー。

以下のトピックで、*nnn* は複数のコンテナーが存在する可能性があることを示しています。コンテナーは、001 から *nnn* (返されるこのタイプのコンテナー数) まで番号が付けられます。1 つのタイプのコンテナーの数として 999 より多い個数はサポートされません。それらのコンテナーに関連する SAML アサーションのデータは無視されます。DSECT にマップされないコンテナーは、可変長になります。

DFHSAML-AnnnVmmm コンテナー**:**

DFHSAML-AnnnVmmm は DATATYPE(CHAR) のコンテナーです。属性 *nnn* の属性値 *mmm* が入ります (*nnn* と *mmm* は 3 桁の数値)。

属性の数は DFHSAML-COUNTS コンテナー内の SAMLC-ATTRNUM です。

この属性の値の数は DFHSAML-ATTRAnnn で示されます。

DFHSAML-ASSQNAME コンテナー**:**

DFHSAML-ASSQNAME は DATATYPE(CHAR) のコンテナーです。 SAML アサーション名前空間が入ります。

可能な値は以下のとおりです。

SAML 1.1
urn:oasis:names:tc:SAML:1.0:assertion

SAML 2.0
urn:oasis:names:tc:SAML:2.0:assertion

このアサーションは URI である必要があります。アサーションがより複雑な場合、3 つに分けて抽出されます。

DFHSAML-ATTRAnnn コンテナー**:**

DFHSAML-ATTRAnnn は DATATYPE(BIT) のコンテナーです。属性 *nnn* の値の数を持つ BIN(31) フィールドが入ります。値の最大数は 999 です。

属性の数は DFHSAML-COUNTS コンテナー内の SAMLC-ATTRNUM です。

DFHSAML-ATTRFnnn コンテナー**:**

DFHSAML-ATTRFnnn は DATATYPE(CHAR) のコンテナーです。 属性 *nnn* の属性名形式が入ります (*nnn* は 3 桁の数値)。

属性の数は DFHSAML-COUNTS コンテナー内の SAMLC-ATTRNUM です。

DFHSAML-ATTRNnnn コンテナー**:**

DFHSAML-ATTRNnnn は DATATYPE(CHAR) のコンテナーです。属性 *nnn* の属性名が入ります (*nnn* は 3 桁の数値)。

属性の数は DFHSAML-COUNTS コンテナー内の SAMLC-ATTRNUM です。

DFHSAML-ATTRSnnn コンテナー**:**

DFHSAML-ATTRSnnn は DATATYPE(CHAR) のコンテナーです。属性 *nnn* の属性名前空間が入ります (*nnn* は 3 桁の数値)。

属性の数は DFHSAML-COUNTS コンテナー内の SAMLC-ATTRNUM です。

DFHSAML-ATTRYnnn コンテナー**:**

DFHSAML-ATTRYnnn は DATATYPE(CHAR) のコンテナーです。属性 *nnn* の属性フレンドリー名が入ります (*nnn* は 3 桁の数値)。

属性の数は DFHSAML-COUNTS コンテナー内の SAMLC-ATTRNUM です。

DFHSAML-AUDNRnnn コンテナー**:**

DFHSAML-AUDNRnnn は DATATYPE(CHAR) のコンテナーです。AudienceRestriction 名が入ります。

返されるコンテナーの数は AUDNRNUM です。

DFHSAML-AUTHMETH コンテナー**:**

DFHSAML-AUTHMETH は DATATYPE(CHAR) のコンテナーです。トークン所有者を認証するために使用されるメソッドが入ります。

メソッドには password、Kerberos、および ltpa などがあります。

DFHSAML-CERTIDN コンテナー**:**

DFHSAML-CERTIDN は DATATYPE(CHAR) のコンテナーです。SAML 署名者の X.509 証明書に対する発行者の識別名が入ります。

DFHSAML-CERTSDN コンテナー**:**

DFHSAML-CERTSDN は DATATYPE(CHAR) のコンテナーです。SAML 署名者の X.509 証明書に対するサブジェクトの識別名が入ります。

DFHSAML-CERTSNUM コンテナー**:**

DFHSAML-CERTSNUM は DATATYPE(CHAR) のコンテナーです。SAML 署名者の X.509 証明書シリアル番号が入る 8 桁のフィールドです。

DFHSAML-CONFMETH コンテナー**:**

DFHSAML-CONFMETH は DATATYPE(CHAR) のコンテナーです。この SAML トークンで使用される SubjectConfirmation メソッドが入ります。

有効なメソッドは holder-of-key、bearer、または sender-vouches です。返されるストリングは、OASIS SAML トークン・プロファイル 1.1 および 2.0 に基づきます。

注**:** 複数の確認メソッドを使用する SAML トークンは、サポートされません。複数の確認メソッドが存在する場合、結果は予測不能です。

DFHSAML-COUNTS コンテナー**:**

DFHSAML-COUNTS は DATATYPE(BIT) のコンテナーです。可変長コンテナー出力の数が入ります。

DFHSAML-FLAGS コンテナー**:**

DFHSAML-FLAGS は DATATYPE(CHAR) のコンテナーです。フラグ・バイトのコレクションが入ります。

DFHSAML-ISSUER コンテナー**:**

DFHSAML-ISSUER は DATATYPE(CHAR) のコンテナーです。発行者の名前が入ります。

DFHSAML-NAMID コンテナー**:**

DFHSAML-NAMID は DATATYPE(CHAR) のコンテナーです。名前フォーマット・プロパティーの値が入ります。

DFHSAML-NAMIDF コンテナー**:**

DFHSAML-NAMIDF は DATATYPE(CHAR) のコンテナーです。ストリング・ベースの ID 情報の分類を表す URI 参照が入ります。

DFHSAML-NAMIDQ コンテナー**:**

DFHSAML-NAMIDQ は DATATYPE(CHAR) のコンテナーです。名前を修飾するセキュリティーまたは管理可能ドメインが入ります。

DFHSAML-NAMIDSP コンテナー**:**

DFHSAML-NAMIDSP は DATATYPE(CHAR) のコンテナーです。サービス・プロバイダーまたはエンティティーのプロバイダーの所属によって確定される名前 ID が入ります。

DFHSAML-NAMIDSPQ コンテナー**:**

DFHSAML-NAMIDSPQ は DATATYPE(CHAR) のコンテナーです。サービス・プロバイダーの名前またはプロバイダーの所属が入ります。

DFHSAML-OUTTOKEN コンテナー**:**

DFHSAML-OUTTOKEN は DATATYPE(CHAR) のコンテナーです。SAML トークンが入ります。

入力コンテナーである場合は、別のサービス・プロバイダーに経路指定される (または拡張された後に経路指定される) 検証済みトークンが入ります。

出力コンテナーである場合は、DFHSAML 処理によって出力される SAML トークンが入ります。処理が検証または抽出である場合、このトークンは検証済み、抽出済み、変更済み、または破棄済みのトークンです。

DFHSAML-PROXYnnn コンテナー**:**

DFHSAML-PROXYnnn は DATATYPE(CHAR) のコンテナーです。ProxyRestriction Audience 名が入ります。

DFHSAML-RESPONSE コンテナー**:**

DFHSAML-RESPONSE は DATATYPE(BIT) のコンテナーです。内部で使用される応答コードが入ります。

DFHSAML-SAMLID コンテナー**:**

DFHSAML-SAMLID は DATATYPE(CHAR) のコンテナーです。SAML 2.0 の ID または SAML 1.1 の AssertionID を表すストリングが入ります。

DFHSAML-SUBJADDR コンテナー**:**

DFHSAML-SUBJADDR は DATATYPE(CHAR) のコンテナーです。SubjectLocality の IP アドレスが入ります。

制約事項**:** SAML 2.0 では、このコンテナーは返されません。

DFHSAML-SUBJDNS コンテナー**:**

DFHSAML-SUBJDNS は DATATYPE(CHAR) のコンテナーです。SubjectLocality の DNSAddress が入ります。

DFHSAML-TIMES コンテナー**:**

DFHSAML-TIMES は DATATYPE(CHAR) のコンテナーです。時間値のコレクションが入ります。

CICS によって生成されるコンテナー

CICS は、変数配列や長ストリングなどのデータを保管するためのコンテナーを生成します。 これらのコンテナーはパイプライン処理中に作成され、アプリケーション・プログラムとの間の入出力として使用されます。 これらのコンテナーの接頭部は DFH です。

これらのコンテナーの命名規則は、コンテナー名を要求内で固有にするためにそれらを数値接尾部と組み合わせて作成した CICS モジュールを使用することです。 パイプライン処理中に作成されるコンテナー名は次のとおりです。

DFHPIAXIS-*nnnnnn*
: ストリングと変数配列を保管するために使用されるコンテナー。Axis2 パイプラインのアプリケーションに渡されます。 このコンテナーはバイナリー・データを含めることもできます。

DFHPICC-*nnnnnnnn*
: ストリングと変数配列を保管するために使用されるコンテナー。アプリケーションに渡されます。 このコンテナーはバイナリー・データを含めることもできます。

DFHPIII-*nnnnnnnn*
: パイプラインが MTOM メッセージ・ハンドラーで使用可能であり、直接モードで実行中の場合に作成される、アウトバウンド接続コンテナー。 これらのコンテナーは、アプリケーション・プログラムによってバイナリー・データがコンテナーではなくフィールドに提供されている場合に作成されます。

DFHPIMM-*nnnnnnnn*
: MIME メッセージの処理中に作成されるインバウンド接続コンテナー。 これらのコンテナーは、MTOM メッセージ・ハンドラーがパイプラインで使用可能な場合に、CICS によって生成されます。 直接モード処理が有効な場合、これらのコンテナーはアプリケーションに直接移動する場合があります。

DFHPIXO-*nnnnnnnn*
: パイプラインが MTOM メッセージ・ハンドラーで使用可能であり、互換モードで実行中の場合に作成される、アウトバウンド接続コンテナー。

番号が付いたコンテナー名は、Web サービス要求ごとに 1 から始まります (例えば、DFHPICC-00000001)。 ただし、アプリケーション・プログラムが **INVOKE SERVICE** コマンドを使って同じチャネルで複数の Web サービス要求を開始する場合、1 つの応答のためにアプリケーションに戻されたコンテナーが、それ以降の要求でも引き続き存在する可能性があります。 このような状況では、CICS は、コンテナーがすでに存在しているかどうかを確認し、生成されるコンテナーの数を増やして命名の競合を回避します。

ユーザー・コンテナー

このコンテナーは、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。

Web サービスのための実行時処理

Web サービス・プロバイダーに要求を送信したり、Web サービス・リクエスターから要求を受信したりするには、アプリケーション (またはラッパー・プログラム) が CICS の Web サービス・サポートと正しく対話する必要があります。また、インバウンド/アウトバウンド要求の処理方法を決定するためにパイプラインで実行される処理を制御することもできます。

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。

JCL プロシージャー 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** パラメーターでコンテナーの名前を指定できます。

- 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。

  API コマンドをコーディングしてコンテナーで作業する場合に、CHANNEL オプションを指定する必要はありません。コンテナーはすべて現在のチャネル (プログラムに渡されたチャネル) に関連付けられているためです。 チャネルの名前を知りたい場合は、**EXEC CICS ASSIGN CHANNEL** コマンドを使用します。

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。

アプリケーション・プログラムが応答メッセージを出す際にエラーになる場合は、アプリケーションが同期点を実行していない限り、CICS が変更をすべてロールバックします。

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。

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 SERVICE** 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。このサービス呼び出し方式は JSON ではサポートされていません。

手順

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。 DFHWS-DATA 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。 以下のコマンドを使用します。

   ```
   EXEC CICS INVOKE SERVICE(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。

   詳しくは、 213 ページの『Web サービスのためのローカル最適化』を参照してください。
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。
4. サービス・リクエスターが、呼び出される Web サービスから SOAP 障害メッセージを受け取る場合は、アプリケーション・プログラムで変更をロールバックすべきかを決定する必要があります。 SOAP 障害の場合、RESP2 値が 6 である INVREQ エラーがアプリケーション・プログラムに戻されます。 ただし、最適化が有効であれば、リクエスターとプロバイダーの両方で同じトランザクションが使用されます。 ローカルで最適化された Web サービス・プロバイダーでエラーが起こる場合は、このトランザクションによって行われたすべての処理

が、プロバイダーとリクエスターの両方でロールバックされます。 RESP2 値が 16 である INVREQ エラーがリクエスターに戻されます。

Web サービスのためのローカル最適化

Web サービス要求のローカル最適化を可能にするために、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・ファイルでプロバイダー・アプリケーション名を使用できます。

INVOKE SERVICE コマンドを使用して、URIMAP(urimap) または URI(uri) を指定できます (uri は呼び出される Web サービスの URI)。URIMAP を指定した場合、CICS は指定されたクライアント・モード URIMAP を使って URI を解決します。これらのオプションが省略される場合、CICS は WEBSERVICE の生成元である Web サービス記述 (WSDL) で指定された URI を使用します。

指定された WEBSERVICE がリクエスター・モード PIPELINE に配置されている場合、CICS はリモート Web サービスを呼び出します。これが最も標準的なシナリオです。

指定された WEBSERVICE がプロバイダー・モード PIPELINE に配置されている場合、CICS はローカルにサービスを呼び出します。この最適化を使用すると、EXEC CICS INVOKE SERVICE コマンドが EXEC CICS LINK コマンドに最適化されます。 結果としてパフォーマンス上の大きな利点が得られますが、以下の制限事項も生じます。

- PIPELINE は駆動されないため、ハンドラー・プログラムは使用されません。
- チャネル上に PIPELINE 制御コンテナーが存在しません。DFHWS-DATA、DFHWS-OPERATION、DFHWS-URI コンテナーなど、いくつかのコンテナーは存在します。 XML を通常含むコンテナーはまったく存在しません (これには DFH-REQUEST、DFHWS-BODY、DFHWS-XMLNS コンテナーが含まれます)。
- プロバイダーとリクエスターの両方のアプリケーションは同じコピーブックを共用する必要があり、同じプログラミング言語で実装される必要があります。
- プロバイダーとリクエスターの両方のアプリケーションは 1 つの作業単位を共用します。
- Web サービスが応答を戻すと予期されない場合、ターゲット PROGRAM が完了してしまうまで、EXEC CICS INVOKE SERVICE コマンドはアプリケーションに制御を戻しません。

ローカルに最適化された Web サービスを使用する場合、PIPELINE を介してデータを処理する必要があるならば、 218 ページの『リクエスター・パイプライン処理を制御するためのオプション』で説明されている cics URI 形式を使用してください。このメカニズムは、完全に最適化された手法を使う場合に比べると非効率的ですが、ネットワークの使用による処理コストを防ぎます。

Web サービス・アシスタントによって生成されるコードの実行時の制限

CICS は実行時に、Web サービス記述 (WSDL) に準拠する有効な SOAP メッセージのほとんどすべてを、同等のデータ構造に変換できます。ただし、サービス・リ

クエスターまたはサービス・プロバイダーのアプリケーションを、Web サービス・アシスタントのバッチ・ジョブを使用して開発する際は、いくつかの制限があります。

コード・ページ

CICS は、コード・ページを識別する適切な HTTP または WebSphere MQ ヘッダーがあれば、どのようなコード・ページで送信される SOAP メッセージでもサポートします。 CICS は、アプリケーション・プログラムで必要とされるコード・ページに変換する前に、パイプラインで処理するために、SOAP メッセージを UTF-8 に変換します。 パフォーマンス・インパクトを最小化するには、SOAP メッセージを CICS に送信する際に、UTF-8 コード・ページの使用をお勧めします。 CICS では常に SOAP メッセージを UTF-8 で送信します。

CICS は、アプリケーション・データが EBCDIC または UTF-16 でエンコードされている場合にのみ、SOAP メッセージを変換できます。UTF-8、ASCII および ISO8859-1 などのコード・ページでデータがエンコードされることを予期するアプリケーションはサポートされません。 データ構造および SOAP メッセージで DBCS 文字を使用したい場合は、DBCS をサポートするコード・ページを指定する必要があります。 ユーザーが選択する EBCDIC コード・ページは、Java および z/OS 両方の変換サービスによってもサポートされていなければなりません。 また、z/OS 変換サービスは、 SOAP メッセージのコード・ページから UTF-8 への変換をサポートするように構成されている必要があります。UTF-16 のサポートについて詳しくは、アプリケーション・データでの UTF-16 サポートを参照してください。

適切なコード・ページを設定するには、**LOCALCCSID** システム初期設定パラメーターを使用するか、Web サービス・アシスタントのジョブでオプションの **CCSID** パラメーターを使用します。 **CCSID** パラメーターを使用する場合、ユーザーが指定する値が、その特定の Web サービスについての **LOCALCCSID** コード・ページを指定変更します。 **CCSID** パラメーターを指定しない場合は、データの変換には **LOCALCCSID** コード・ページが使用され、Web サービス・バインディング・ファイルが US EBCDIC (Cp037) でエンコードされます。

コンテナー

サービス・プロバイダー・モードでは、**PGMINT** パラメーターに値 CHANNEL を指定する場合、アプリケーション・データを保持するコンテナーがバイナリー・モードで書き込みおよび読み取りを行う必要があります。 このコンテナーはデフォルトでは DFHWS-DATA です。 **PUT CONTAINER** コマンドで DATATYPE オプションを BIT に設定するか、FROMCCSID オプションを除外して BIT をデフォルトのままにする必要があります。 例えば次のコードは、現在のチャネルにあるコンテナー CUSTOMER-RECORD 内にあるデータを、バイナリー・モードで書き込む必要があることを明確に示しています。

```
EXEC CICS PUT CONTAINER (CUSTOMER-RECORD)
              FROM (CREC)
              DATATYPE(BIT)
```

コンテナー自身がすべて BIT モードであっても、このデータをマップする言語構造内のすべてのテキスト・フィールドで、EBCDIC コード・ページ (**LOCALCCSID** また

は **CCSID** パラメーターに指定したものと同じコード・ページ) を使用する必要があります。 Web サービス・バインディング・ファイルの生成に DFHWS2LS を使用する場合は、完全なデータ構造の一部を保持するコンテナーがチャネル上に多数あるかもしれません。 この場合、これらの各コンテナーのテキスト・フィールドは、同じコード・ページを使用して読み取りおよび書き込みを行う必要があります。

アプリケーション・プログラムが、SOAP メッセージに変換されるコンテナーを組み込む場合は、アプリケーションがコンテナーに適切な量のデータが含まれるか判断します。 コンテナーが保持するデータが予想より少ない場合は、CICS が変換エラーを出します。

アプリケーション・プログラムが **INVOKE SERVICE** コマンドを使用する場合、CICS に渡される任意のコンテナーが再利用され、その中のデータが置き換えられる可能性があります。 これらのコンテナー内のデータを保持したい場合には、プログラムを実行する前に、新しいチャネルを作成してコンテナーをそれにコピーしてください。 リクエスター・モードの Web サービスでもあるプロバイダー・モードの Web サービスがある場合は、**INVOKE SERVICE** コマンドを使用する際に、最初に接続されていたデフォルトのチャネルを使用するのではなく、別のチャネルを使用することをお勧めします。 アプリケーション・プログラムが **INVOKE SERVICE** コマンドを何度も使用する場合は、CICS を呼び出すたびに異なるチャネルを使用するか、最初の要求における重要なデータをすべて保管してから、2 番目の要求を行うことをお勧めします。

Web サービス記述への準拠

Web サービス記述では、オプションで、考えられる SOAP メッセージの内容の一部を記述できます。 この場合、DFHWS2LS が、生成された言語構造内でフィールドを割り振り、内容があるかどうかを示します。 CICS は実行時にこれらのフィールドを適宜設定します。 例えば存在フラグまたはオカレンス・フィールドなどのフィールドが、情報がないことを示す場合、アプリケーション・プログラムは、そのオプションの内容に関連付けられたフィールドを処理しません。

CICS が SOAP メッセージを変換する際に、SOAP メッセージの内容が一部欠落している場合、データ構造内のこれに相当するフィールドは、アプリケーション・プログラムに渡されるときに初期化されません。

Web サービス記述では、SOAP メッセージを読み取る際に使用する、空白文字の処理規則も指定できます。 CICS は実行時にこの規則を実装します。

- `xsd:whiteSpace` ファセットの値が `replace` である場合、「タブ」および「復帰」などの空白文字はスペースで置き換えられます。
- `xsd:whiteSpace` ファセットの値が `collapse` である場合、末尾の空白文字は、SOAP メッセージを生成する際に除去されます。 実行時に、XML スキーマの仕様に従ってインバウンド SOAP メッセージが解析され、すべての先行、末尾、および組み込みの空白が除去されます。

SOAP メッセージ

CICS は、SOAP メッセージの内容の導出をサポートしていません。 例えば、SOAP メッセージは、`xsi:type` 属性を使用してエレメントに特定のタイプを指定し、併せて `xsi:schemaLocation` 属性でエレメントを記述するスキーマのロケーシ

ョンを指定できます。 CICS は、動的にスキーマを取得し、スキーマの内容に基づいてエレメントの値を変換する機能をサポートしていません。 CICS は、Web サービス・アシスタントに設定されたマッピング・レベルが 1.1 以上であるときに xsi:nil 属性をサポートしますが、これはサポートされる唯一の XML スキーマ・インスタンス属性です。

DFHWS2LS は、SOAP メッセージ内のいくつかの値について、最大長や最大サイズを推測する必要があるかもしれません。 例えば、XML スキーマが xsd:string の最大長を指定しない場合、DFHWS2LS は最大長を 255 文字と推測し、これに応じて言語構造を生成します。 この値は、DFHWS2LS で **DEFAULT-CHAR-MAXLENGTH** パラメーターを使用して変更できます。 CICS が実行時に、言語構成に割り振られたスペースより大きい値を持つ SOAP メッセージを検出すると、CICS は変換エラーを出します。

CICS がサービス・プロバイダーである場合は、SOAP 障害メッセージがリクエスターに戻されます。 CICS がサービス・リクエスターである場合は、適切な RESP2 コードが **INVOKE SERVICE** コマンドから戻されます。

XML では、< および > 文字のように、特殊な意味を持つ文字があります。実行時に CICS が処理する文字配列内にこのような特殊文字が出現する場合は、同等のエンティティーで置き換えられます。 サポートされる XML エンティティーは、次のとおりです。

文字	**XML** エンティティー
&	&
<	<
>	>
"	"
'	'

CICS は、空白文字コードに使用される数字参照の正規形式もサポートします。

文字	**XML** エンティティー
タブ	
復帰
改行

このサポートは、呼び出されるどのパイプライン・ハンドラー・プログラムにも拡張されないことに注意してください。

ヌル文字 (x'00') は、どの XML 文書でも無効です。 アプリケーション・プログラムに備えられた文字タイプ・フィールドがヌル文字を含む場合は、CICS がその時点でデータを切り捨てます。 このため、文字配列をヌル終了ストリングとして処理できます。 DFHWS2LS によって base64Binary または hexBinary の XML スキーマのデータ・タイプから生成される文字タイプのフィールドは、バイナリー・データを表示します。このフィールドは、ヌル文字を切り捨てずに含むことがあります。

重要: CICS は、構造化されたアプリケーション・データから XML および JSON データを生成します。そのアプリケーション・データに、事前フォーマットされた XML、JSON、HTML、JPEG イメージ、または他の意味あるコンテンツ・タイプのように見えるビット・パターンが含まれている場合、CICS は、このセマンティックな意味を認識せず、そのようなデータを通常のテキストまたはバイナリー・データとして処理します。CICS は、データ内のパターンを認識したり、またはエンコードされたデータを異なる方法で処理しようとしたりはしません。例えば、データに事前フォーマットされた XML (CDATA でエンコードされたテキストなど) が含まれている場合、そのデータは他のデータと同じ方法で処理されます。アプリケーション・データ `"An example: <here>"` を考慮してください。この例のアプリケーション提供のデータには、XML タグのように見えるものが含まれていますが、これはロー・テキストとして処理され、XML 表現 `"An example: < here > "` となります。アプリケーションが XML 自体を生成する必要がある場合には、XML スキーマで `xsd:any` 構文を使用するか、アシスタントで XML-ONLY=TRUE を使用することを考慮してください。

SOAP 障害メッセージ

CICS がサービス・プロバイダーであり、アプリケーション・プログラムで SOAP 障害メッセージを出したい場合は、**SOAPFAULT CREATE** コマンドを使用します。 この API コマンドを使用するには、Web サービス・アシスタントの **PGMINT** パラメーターに値 `CHANNEL` を指定する必要があります。 この値を指定せずに、アプリケーション・プログラムが **SOAPFAULT CREATE** コマンドを呼び出すと、CICS が SOAP 応答メッセージを生成しようとしません。

パイプライン処理のカスタマイズ

独自のメッセージ・ハンドラーを提供することに加えて、一連のグローバル・ユーザー出口点 (GLUE) を使用し、パイプライン内のインバウンドおよびアウトバウンド Web サービスで生じる処理をカスタマイズすることもできます。

始める前に

パイプラインをカスタマイズする前に、グローバル・ユーザー出口プログラムを作成するうえでのベスト・プラクティスを理解しておく必要があります。 サービス・プロバイダー・パイプラインをカスタマイズする場合は、提供されている DFHPITP または Axis2 アプリケーション・ハンドラーをパイプラインで使用する必要があります。

このタスクについて

パイプライン・ドメイン出口を使用すると、Web サービス・プロバイダー・パイプライン、Web サービス・リクエスター・パイプライン、またはセキュリティー・ハンドラーを備えた Web サービス・リクエスター・パイプライン上のコンテナーにアクセスできます。パイプライン・グローバル・ユーザー出口の詳細については、パイプライン・ドメイン出口を参照してください。

手順

1. 次のようにして、使用するグローバル・ユーザー出口ポイントを選択します。

- Web サービス・プロバイダー・パイプラインのコンテナーにアクセスするには、XWSPRRWI、XWSPRROI、XWSPRROO、XWSPRRWO のいずれかの GLUE を使用します。
- Web サービス・リクエスター・パイプラインのコンテナーにアクセスするには、XWSRQRWO、XWSRQROO、XWSROROI、XWSRQRWI のいずれかの GLUE を使用します。
- 保護された Web サービス・リクエスター・パイプラインのコンテナーにアクセスするには、XWSSRRWO、XWSSRROO、XWSSRROI、XWSSRRWI のいずれかの GLUE を使用します。

2. DFH$PIEX サンプル出口プログラムを使用して、独自のグローバル・ユーザー出口プログラムを作成します。 DFH$PIEX は SDFHSAMP ライブラリーの中にあります。プログラムをスレッド・セーフにすることをお勧めします。
3. グローバル・ユーザー出口プログラムを使用可能にします。
4. グローバル・ユーザー出口プログラムをテストして、正しく機能することを確認します。

リクエスター・パイプライン処理を制御するためのオプション

サービス・リクエスター・パイプラインにおいて、メッセージ・ハンドラーは URI を変更することによって Web サービス要求が送られる場所を決定できます。 CICS はさまざまな URI 形式をサポートするため、パイプラインで Web サービス要求が処理される方法をずっと柔軟に制御することができます。

サービス・リクエスター・パイプラインの処理の終わりに達したとき、次のようなオプションの中から選ぶことができます。

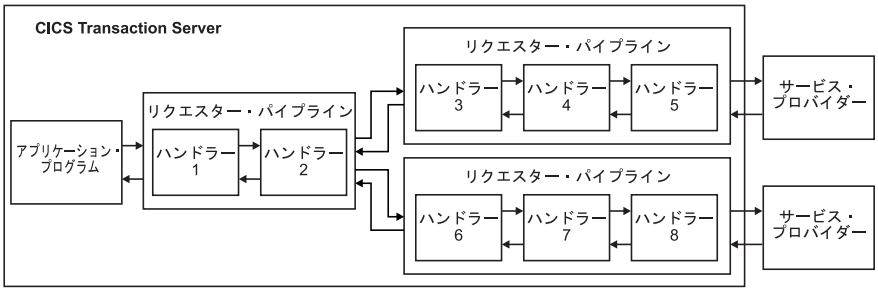
プログラムにリンクする
: `cics://PROGRAM/`*`program`* (*program* はターゲット・アプリケーション・プログラムの名前) という形式に URI を変更した場合、CICS は **`EXEC CICS LINK`** コマンドを使用して現在のチャネルとそのコンテナー (または COMMAREA) をプログラムに渡します。

  この処理は、サービス・リクエスターとサービス・プロバイダー・アプリケーションが同じ CICS 領域に属する場合に行われるローカル最適化に似ています。 しかし、この URI 形式を使用すると、パイプラインと任意のカスタム・メッセージ・ハンドラーを介して最初に要求が実行されるという利点があります。 ターゲット・アプリケーション・プログラムは、コンテナーまたは COMMAREA の内容を処理できる必要があります。

別のリクエスター・モード・パイプラインを開始する
: `cics://PIPELINE/`*`pipeline`*`?targetServiceUri=`*`targetServiceUri`* (*pipeline* は PIPELINE リソースの名前、*targetServiceUri* は DFHWS-URI コンテナーに入れる URI) という形式に URI を変更した場合、CICS は指定されたリクエスター・パイプラインに現在のチャネルとそのコンテナーを渡します。 要求をサービス・プロバイダーに送る前に、複数のリクエスター・パイプラインを互いにリンクするには、この URI を使用してください。 連結できるリクエスター・パイプラインの数には制限がありません。

  以下の例では、1 つの汎用リクエスター・パイプラインが 1 つのアプリケーションをサポートしています。 メッセージ・ハンドラー 1 または 2 は、コンテナー内のアプリケーション・データに応じてそれぞれの要求の

URI を変更することができ、異なるメッセージ・ハンドラーを含む 2 つのリクエスター・パイプラインのいずれかに要求を送ります。



この例は 1 つのサービス・リクエスター・アプリケーションだけを示していますが、多数のアプリケーションで同じ汎用リクエスター・パイプラインを使用して、要求が適切な Web サービス・プロバイダーに最終的に送られる前に、異なるリクエスター・パイプラインに要求を送ることもできます。

プロバイダー・モード・パイプラインに要求を直接送信する

`cics://SERVICE/`*`service`*`?targetServiceUri=`*`targetServiceUri`* (*service* はターゲット・サービスの名前、*targetServiceUri* はサービスのパス) という形式に URI を変更した場合、CICS はパスを URIMAP にマッチングすることによって要求を解決し、正しいプロバイダー・パイプラインに要求を渡します。 ネットワークを使わずにリクエスター・パイプラインとプロバイダー・パイプラインの両方を介して要求を処理したい場合には、このオプションを使用してください。

また、リクエスター・アプリケーションとプロバイダー・アプリケーションが異なる言語で書かれ (または異なるマッピング・レベルを使用し)、異なるバイナリー・データを想定しているような場合にも、この URI が役立つでしょう。

URI を使用したリクエスター・パイプライン処理の制御

サービス・リクエスター・パイプラインにおいて、メッセージ・ハンドラーは URI を変更することにより Web サービス要求をどこに送るかを決定できます。 URI 形式を変更すると、別のリクエスター・パイプラインを開始したり、ネットワークを介して要求を送信せずにサービス・プロバイダー・パイプラインを開始するなどの最適化を実行できます。

始める前に

リクエスター・パイプラインにどのオプションを実装するかを決定します。 詳細については、 218 ページの『リクエスター・パイプライン処理を制御するためのオプション』を参照してください。

このタスクについて

Web サービス・リクエスター・アプリケーションは **EXEC CICS INVOKE SERVICE** コマンドを使って DFHWS-URI コンテナーのデータを設定できます。アプリケーションによって値が提供されない場合、CICS は Web サービス・バインディング・ファイル内の値を使ってコンテナーのデータを設定します。 URI を変更するには、

このコンテナーの内容を変更するメッセージ・ハンドラーを作成する必要があります。

手順

1. 以下のいずれかの URI 形式に従って、DFHWS-URI コンテナーを変更するメッセージ・ハンドラーを作成します。
   - アプリケーション・プログラムにリンクするには、cics://PROGRAM/*program* という URI を使用します (*program* はターゲット・アプリケーション・プログラム)。データ変換は行われないため、アプリケーション・プログラムで現在のチャネル上のコンテナーの内容を処理できるようにする必要があります。アプリケーション・プログラムは、現在のチャネルとコンテナー、あるいは COMMAREA を渡すことができます。
   - ネットワークを通さずにプロバイダー・パイプラインを開始するには、cics://SERVICE/*service*?targetServiceUri=*targetServiceUri* という URI を使用します (*service* はサービスの名前、*targetServiceUri* はサービスのパス)。サービスのパスを使ってトランスポート・ハンドラーは要求を解決する URIMAP リソースを見つけた後、正しいプロバイダー・パイプラインにそれを渡します。 CICS は、その処理ではサービスの名前を使用しません。

     サービス用の URIMAP リソースがインストールされていない場合、エラーが発生します。 URIMAP リソース定義では USAGE(PIPELINE) を指定する必要もあります。 トランスポート・ハンドラーは **targetServiceUri** パラメーターの値を DFHWS-URI コンテナーに入れて、プロバイダー・パイプラインを開始します。
   - 別のリクエスター・パイプラインを開始するには、cics://PIPELINE/*pipeline*?targetServiceUri=*targetServiceUri* という URI を使用します (*pipeline* は開始する PIPELINE リソースの名前、*targetServiceUri* は DFHWS-URI コンテナーで次のパイプラインに渡す値)。

   それぞれのタイプの URI には追加のパラメーターがあり、照会ストリングとしてこれらを追加できます。 これらの URI の形式とコーディング上の規則について、詳しくは、 193 ページの『DFHWS-URI コンテナー』を参照してください。
2. XML エディターを使用して、次のようにメッセージ・ハンドラーをパイプライン構成ファイルに追加します。

```
<service>
  <service_handler_list>
    <handler>
      <program>MYPROG</program>
    </handler>
  </service_handler_list>
</service>
```

3. この新しいメッセージ・ハンドラー・プログラムをパイプラインに含めるために、リクエスター・パイプライン用の PIPELINE リソースを使用不可にし、破棄して、再インストールします。
4. メッセージ・ハンドラー・プログラムを CICS 領域にインストールします。

タスクの結果

リクエスター・パイプラインを通り抜ける次のサービス要求は、新しいメッセージ・ハンドラーによって処理されます。

次のタスク

リクエスター・パイプラインの変更内容をテストすることによって、サービス要求が正しい場所に送られ、メッセージ・ハンドラー・プログラムが設計どおりに動作することを確認します。

Web サービス・トランザクションのサポート

Web Services Atomic Transaction (WS-AtomicTransaction) 仕様および Web Services Coordination (WS-Coordination) 仕様では、複数のトランザクション処理システムを Web サービス環境で相互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。

これらの仕様は、OASIS で参照できます。

注**:** CICS は、2004 年 11 月レベルの仕様をサポートします。

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、これらの仕様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-Coordination 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。

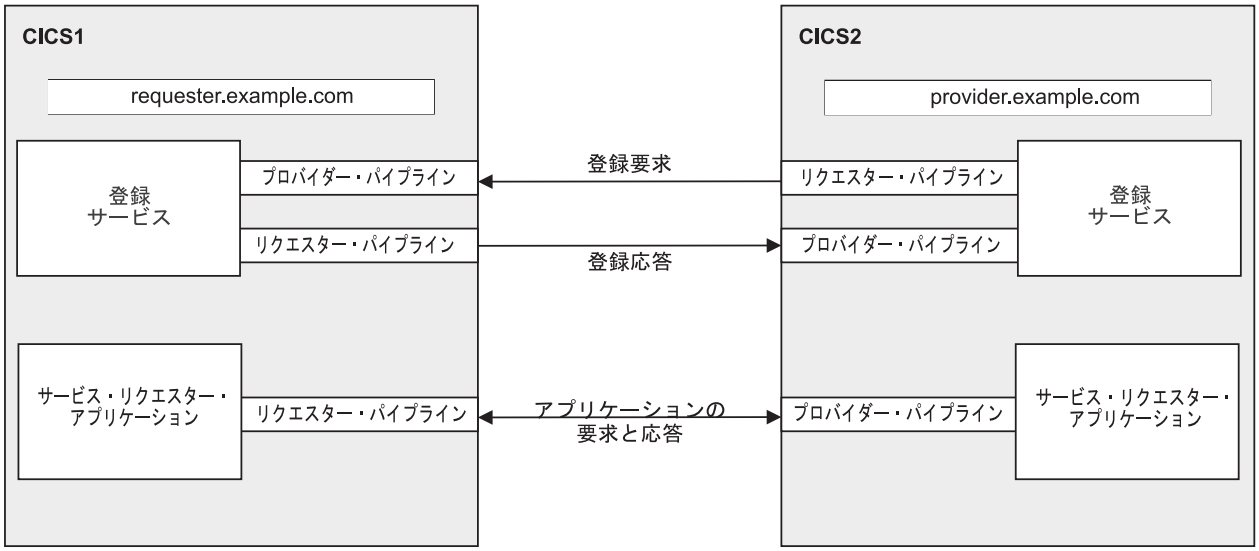


図 *24.* 登録サービス

図 24 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-Coordination 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ともに 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、 223 ページの図 25 に示します。

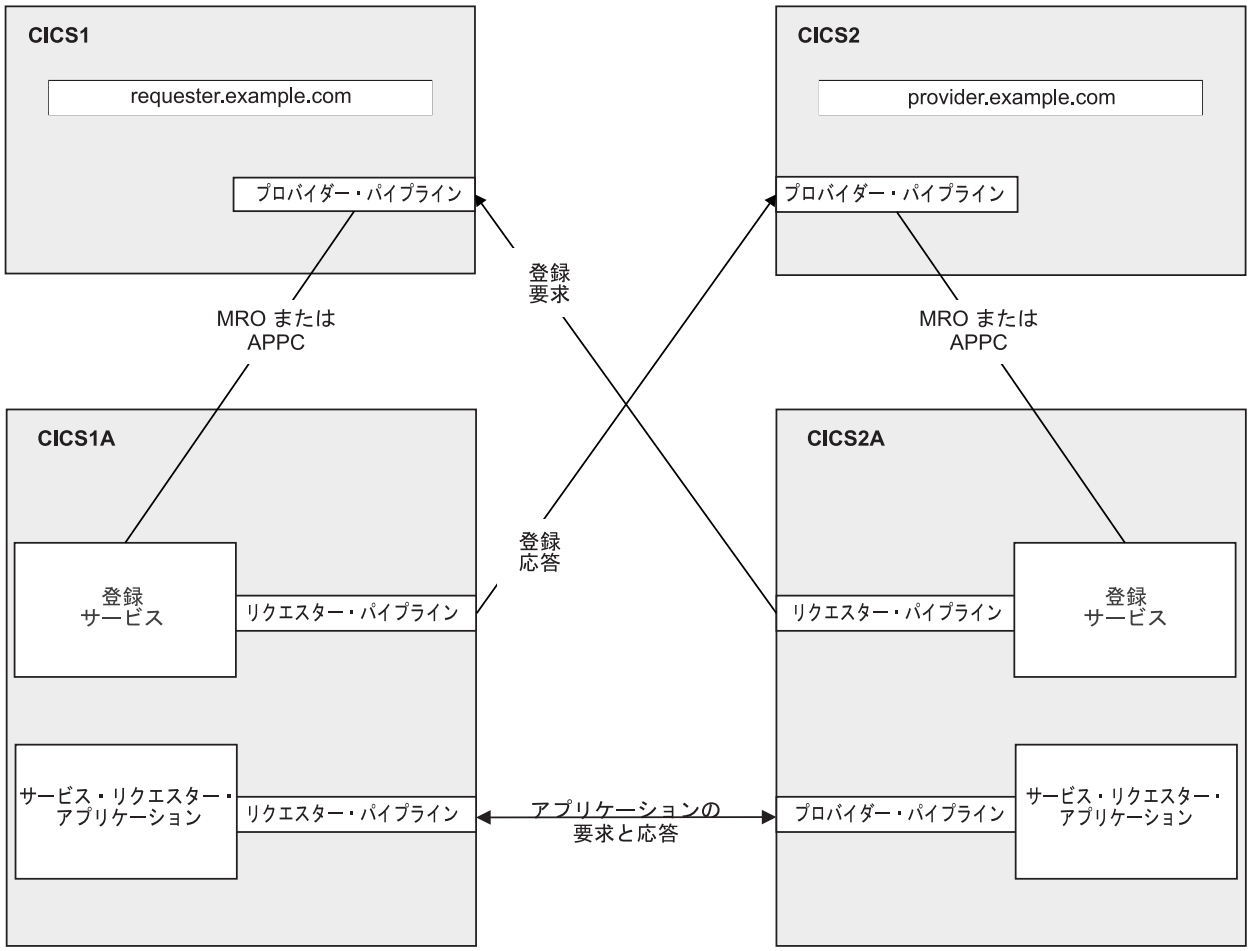


図 *25. CICSPlex* での登録サービス

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 Web サービス・トランザクション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、図 25 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。

Web サービス・トランザクションに合わせた 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 Web サービス・トランザクションの場合、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。

始める前に

これらのリソースをインストールする前に、Web サービス・トランザクションをサポートするための CICS 提供のパイプライン構成ファイルの場所を確認しておく必要があります。 デフォルトでは、構成ファイルは `/usr/lpp/cicsts/cicsts55/pipeline/configs` ディレクトリーにありますが、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。

このタスクについて

Web サービス・トランザクション対応の CICS サポートでは、CICS 提供の登録サービスが使用されます。 登録サービスは、サービス・プロバイダーとサービス・リクエスターで構成されます。 アプリケーションがすべてサービス・プロバイダーであったり、すべてサービス・リクエスターであったりする場合でも、サービス・プロバイダーとサービス・リクエスター両方のリソース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。

Web サービス・トランザクション用に CICS を構成するために必要なリソースはすべて DFHWSAT グループに提供されています。ただし例外として DFHPIDIR は DFHPIVS、DFHPIVR、または DFHPICF グループのいずれかに提供されています。 DFHWSAT グループは DFHLIST リストに含まれないため、自動的にはインストールされません。 CICS によって提供される DFHWSAT グループのリソースを変更することはできません。

Web サービス・トランザクション用に CICS を構成するには、次のようにします。

手順

1. DFHPIDIR データ・セットを始動 JCL に追加します。 DFHPIDIR には、コンテキストとタスクの間のマッピングが格納されます。
   a. DFHPIDIR データ・セット用の新しい DD ステートメントを CICS 始動 JCL に追加します。
   b. DFHDEFDS.JCL の情報を使用して DFHPIDIR データ・セットを作成します。 DFHPIDIR のデフォルトの RECORDSIZE は 1 KB で、ほとんどの用途ではこれが適しています。 必要に応じて、より大きい RECORDSIZE を使って DFHPIDIR を作成することができます。
   c. データ・セット用の適切なグループ (DFHPIVS、DFHPIVR、または DFHPICF) を CICS システムにインストールします。 これらのグループについて詳しくは、WS-AT データ・セットの定義を参照してください。

複数の CICS 領域の間で DFHPIDIR ファイルを共用するには、MRO を介してそれらの領域が論理的に接続されている必要があります。 論理サーバーとして機能している領域のグループにつき 1 つのデータ・セットをインストールする必要があります。

ヒント: 論理的に接続されていない領域間でのデータ・セットの共用は勧められていません。

2. DFHWSAT グループの内容を別のグループにコピーします。 CICS によって提供される DFHWSAT グループの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。
3. 登録サービスのプロバイダー 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/cicsts55/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。

   パイプライン構成ファイルは、提供されているまま使用して、内容を変更しないでください。
4. 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。
5. 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。
6. 登録サービスのリクエスター 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/cicsts55/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。

   パイプライン構成ファイルは、提供されているまま使用して、内容を変更しないでください。
7. PIPELINE リソースをインストールします。 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。
8. 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。

9. 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。

タスクの結果

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 プロトコルと WS-Coordination 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。

次のタスク

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。

Web サービス・トランザクション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、Web サービス・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> エレメントおよび <service_parameter_list> エレメントを指定する必要があります。

始める前に

サービス・プロバイダー・アプリケーションを Web サービス・トランザクションに参加させたい場合は、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セットアップされていると、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 Web サービス・トランザクションにのみ参加します。

このタスクについて

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 Web サービス・トランザクション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たパイプライン構成ファイルの例がファイル /usr/lpp/cicsts/cicsts55/samples/pipelines/wsatprovider.xml (/usr/lpp/cicsts/cicsts55 は z/OS UNIX 上の CICS ファイルのデフォルト・インストール・ディレクトリー) で提供されています。

手順

1. 端末ハンドラーの定義で、<headerprogram> エレメントを <cics_soap_1.1_handler>、<cics_soap_1.2_handler>、<cics_soap_1.1_handler_java>、または <cics_soap_1.2_handler_java> 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。

```
<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で必要とされる場合には、それらも指定してください。

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>
http://address:port/cicswsat/RegistrationService
</registration_service_endpoint>
```

address は、登録サービス・プロバイダー・パイプラインがある 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。

```
<registration_service_endpoint>
http://provider.example.com:7160/cicswsat/RegistrationService
</registration_service_endpoint>
```

Web サービス・トランザクションに合わせたサービス・リクエスターの構成

サービス・リクエスター・アプリケーションの目的が、Web サービス・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> エレメントおよび <service_parameter_list> エレメントを指定する必要があります。

始める前に

サービス・リクエスター・アプリケーションを Web サービス・トランザクションに参加させたい場合は、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セットアップされていると、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 Web サービス・トランザクションにのみ参加します。

このタスクについて

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 Web サービス・トランザクション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たパイプライン構成ファイルの例がファイル /usr/lpp/cicsts/cicsts55/samples/pipelines/wsatrequester.xml (/usr/lpp/cicsts/cicsts55 は z/OS UNIX 上の CICS ファイルのデフォルト・インストール・ディレクトリー) で提供されています。

手順

1. <headerprogram> エレメントを <cics_soap_1.1_handler> エレメント、<cics_soap_1.2_handler> エレメント、<cics_soap_1.1_handler_java> エレメント、または <cics_soap_1.2_handler_java> 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。

   ```
   <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>
   http://address:port/cicswsat/RegistrationService
   </registration_service_endpoint>
   ```

   address は、登録サービス・プロバイダー・パイプラインがある 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。

   <registration_service_endpoint> エレメントの始まり、その内容、および <registration_service_endpoint> エレメントの終わりの間にスペースが存在してはなりません。この例では、分かりやすくするためにスペースを含めています。

3. 同じ作業単位内の複数の要求で同じトランザクション・コンテキストを使用する代わりに、要求ごとに CICS で新しいトランザクション・コンテキストを作成したい場合には、次のようにして <service_parameter_list> 内の空のエレメント <new_tx_context_required/> をパイプライン構成ファイルに追加します。

```
<service_parameter_list>
  <registration_service_endpoint>
  http://requester.example.com:7159/cicswsat/RegistrationService
  </registration_service_endpoint>
  <new_tx_context_required/>
</service_parameter_list>
```

<registration_service_endpoint> エレメントの始まり、その内容、および <registration_service_endpoint> エレメントの終わりの間にスペースが存在してはなりません。この例では、分かりやすくするためにスペースを含めています。

<new_tx_context_required/> 設定は CICS のデフォルトではなく、サンプル・パイプライン構成ファイル wsatprovider.xml には含まれていません。<service_parameter_list> 内の <new_tx_context_required/> をパイプライン構成ファイルに追加する場合、CICS のループバック呼び出しが可能です。この状態ではデッドロックが発生する可能性があることに注意してください。

SOAP メッセージがアトミック・トランザクションの一部であるかどうかの判別

CICS Web サービスがアトミック・トランザクション・パイプラインで呼び出されるときは、SOAP メッセージは必ずしもアトミック・トランザクションの一部である必要はありません。

このタスクについて

SOAP メッセージがアトミック・トランザクションの一部である場合、<soapenv:Header> エレメントには特定の情報が格納されています。 SOAP メッセージがアトミック・トランザクションの一部であるかどうかを調べるには、次のいずれかを行うことができます。

手順

- トレースを使用して <soapenv:Header> エレメントの内容を調べます。
  1. コンポーネント PI を使用して補助トレースを実行し、トレース・レベルを 2 に設定します。
  2. トレース・ポイント PI 0A31 を探します。ここには、要求コンテナーに関する情報が格納されています。 特に、<cicswsa:Action> エレメントの直前に現れる PIIS EVENT - REQUEST_CNT を探します。
- DFHREQUEST コンテナーにデータ RECEIVE-REQUEST が格納されている場合は、DFHWSATP パイプラインでユーザー作成のメッセージ・ハンドラー・プログラムを使用して、このコンテナーの内容を表示します。 この方法を選択する場合は、パイプライン構成ファイルで忘れずにメッセージ・ハンドラー・プログラムを定義します。

例

以下の例は、SOAP エンベロープ・ヘッダーに表示されるアトミック・トランザクションに関する情報を示しています。

```
<soapenv:Header>
  <wscoor:CoordinationContext soapenv:mustUnderstand="1"> 1
    <wscoor:Expires>500</wscoor:Expires>
```

```
    <wscoor:Identifier>com.ibm.ws.wstx:
       0000010a2b5008c80000000200000019a75aab901a1758a4e40e2731c61192a10ad6e921
    </wscoor:Identifier>
    <wscoor:CoordinationType>http://schemas.xmlsoap.org/ws/2004/10/wsat</wscoor:CoordinationType> 2
    <wscoor:RegistrationService 3
       xmlns:wscoor="http://schemas.xmlsoap.org/ws/2004/10/wscoor">
       <cicswsa:Address xmlns:cicswsa="http://schemas.xmlsoap.org/ws/2004/08/addressing">
          http://clientIPaddress:clientPort/_IBMSYSAPP/wscoor/services/RegistrationCoordinatorPort
       </cicswsa:Address>
       <cicswsa:ReferenceProperties
          xmlns:cicswsa="http://schemas.xmlsoap.org/ws/2004/08/addressing">
              <websphere-wsat:txID
                    xmlns:websphere-wsat="http://wstx.Transaction.ws.ibm.com/extension">com.ibm.ws.wstx:
                    0000010a2b5008c80000000200000019a75aab901a1758a4e40e2731c61192a10ad6e921
              </websphere-wsat:txID>
              <websphere-wsat:instanceID
                    xmlns:websphere-wsat="http://wstx.Transaction.ws.ibm.com/extension">com.ibm.ws.wstx:
                    0000010a2b5008c80000000200000019a75aab901a1758a4e40e2731c61192a10ad6e921
              </websphere-wsat:instanceID>
        </cicswsa:ReferenceProperties>
     </wscoor:RegistrationService>
  </wscoor:CoordinationContext>
</soapenv:Header>
```

1. CoordinationContext は、SOAP メッセージがアトミック・トランザクションに参加することを意図していることを示しています。 ここには、調整サービスの一部になる Web サービス・プロバイダーに必要な情報が格納されており、プロバイダーはヘッダーを認識して処理するよう構成されていると仮定されています。
2. CoordinationType は、調整コンテキストが準拠する WS-AT 仕様のバージョンを示します。
3. 調整の RegistrationService は、コーディネーターの登録ポイントの場所、および参加している Web サービスがアトミック・トランザクションのコンポーネントとして登録しようとするときにコーディネーターに戻す必要がある情報を記述します。

アトミック・トランザクションの進行の確認

CICS Web サービスがアトミック・トランザクションの一部として呼び出されると、トランザクションはいくつかの状態を移動します。 これらの状態は、トランザクションが正常化、ロールバックしなければならないかを示します。

このタスクについて

この情報にアクセスしなければならない場合は、以下のいずれかを行うことができます。

手順

- トレースを使用して <cicswsa:Action> エレメントの内容を調べます。
  1. コンポーネント PI を使用して補助トレースを実行し、トレース・レベルを 2 に設定します。
  2. トレース・ポイント PI 0A31 を探します。ここには、要求コンテナーに関する情報が格納されています。 特に、<cicswsa:Action> エレメントの直前に現れる PIIS EVENT - REQUEST_CNT を探します。
- DFHWSATR パイプラインと DFHWSATP パイプラインでユーザー作成のメッセージ・ハンドラー・プログラムを使用して、DFHWS-SOAPACTION コンテナ

ーの内容を表示します。 この方法を選択する場合は、パイプライン構成ファイルで忘れずにメッセージ・ハンドラー・プログラムを定義します。

例

正常に完了して、コミットされるトランザクションの状態は、次のとおりです。

```
"http://schemas.xmlsoap.org/ws/2004/10/wscoor/Register"
"http://schemas.xmlsoap.org/ws/2004/10/wscoor/RegisterResponse"
"http://schemas.xmlsoap.org/ws/2004/10/wsat/Prepare"
"http://schemas.xmlsoap.org/ws/2004/10/wsat/Prepared"
"http://schemas.xmlsoap.org/ws/2004/10/wsat/Commit"
"http://schemas.xmlsoap.org/ws/2004/10/wsat/Committed "
```

ロールバックされるトランザクションの状態は、次のとおりです。

```
 "http://schemas.xmlsoap.org/ws/2004/10/wscoor/Register"
"http://schemas.xmlsoap.org/ws/2004/10/wscoor/RegisterResponse"
"http://schemas.xmlsoap.org/ws/2004/10/wsat/Rollback"
"http://schemas.xmlsoap.org/ws/2004/10/wsat/Aborted"
```

バイナリー・データの MTOM/XOP 最適化のサポート

標準的な SOAP メッセージでは、バイナリー・オブジェクトは Base64 エンコードされており、メッセージの本文に組み込まれています。これによりサイズが 33% 増加します。 大規模なバイナリー・オブジェクトでは、このようなサイズの増加が伝送時間に重大な影響を与えることがあります。 MTOM/XOP を実装すれば、この問題を解決できます。

SOAP Message Transmission Optimization Mechanism (MTOM) 仕様および XML-binary Optimized Packaging (XOP) 仕様 (MTOM/XOP と呼ばれることが多い) は、SOAP メッセージ内の大規模な base64Binary データ・オブジェクトの伝送を最適化するメソッドを定義します。

- MTOM 仕様は、バイナリー・データを分離し (そうしないと Base64 エンコードされる)、 MIME Multipart/Related メッセージを使用してそれを別のバイナリー添付ファイルに送信することによって、SOAP メッセージの最適化のメソッドを概念的に定義します。 このタイプの MIME メッセージは *MTOM* メッセージ と呼ばれます。データをバイナリー・フォーマットで送信するとサイズが著しく削減されるので、SOAP メッセージの伝送が最適化されます。
- XOP 仕様は、MIME メッセージを含むがこれに限定されるわけではない、パッケージ化フォーマットのバイナリー添付ファイルを使用して、XML メッセージの最適化についての実装を定義します。

トランスポート・プロトコルが WebSphere MQ、HTTP、または HTTPS の場合、CICS はリクエスター・パイプラインとプロバイダー・パイプラインの両方でこれらの仕様をサポートします。 Web サービス・プロバイダーまたはリクエスターとして配置される CICS アプリケーションは、base64Binary データを SOAP メッセージに直接組み込む代わりに、このサポートを使用して MTOM メッセージをバイナリー添付ファイルと一緒に送受信できます。

このサポートは、追加オプションをパイプライン構成ファイルに使用することで構成できます。

MTOM/XOP および SOAP

SOAP メッセージの最適化に MTOM/XOP が使用されるとき、SOAP メッセージは XOP 処理を使用して MIME Multipart/Related メッセージに直列化されます。base64Binary データが SOAP メッセージから抽出され、E メールの添付ファイルのような方法で、MIME メッセージ内の別のバイナリー添付ファイルとしてパッケージされます。

添付ファイルがバイナリー・フォーマットでエンコードされるため、base64Binary データのサイズは著しく削減されます。 次に SOAP メッセージの XML が XOP フォーマットに変換されます。これは、base64Binary データを、URL を使用して関連する MIME 添付ファイルを参照する特殊な <xop:Include> エレメントで置き換えることによって行います。

変更された SOAP メッセージは *XOP* 文書 と呼ばれ、メッセージ内にルート文書を形成します。 XOP 文書とバイナリー添付ファイルが一緒になって *XOP* パッケージ を形成します。 SOAP MTOM 仕様に適用されるとき、 XOP パッケージは MTOM フォーマットの MIME メッセージです。

ルート文書は、MIME メッセージ全体のコンテンツ・タイプ・ヘッダーで Content-ID を参照することで識別します。 コンテンツ・タイプ・ヘッダーの例を示します。

```
Content-Type: Multipart/Related; boundary=MIME_boundary;
 type="application/soap+xml"; start="<claim@insurance.com>"
```

start パラメーターには、XOP 文書の Content-ID が含まれます。 このパラメーターがコンテンツ・タイプ・ヘッダーに含まれていない場合は、MIME メッセージの最初の部分が XOP 文書であると想定されます。

MIME メッセージ内の添付ファイルの順序は重要ではありません。例えば一部のメッセージでは、バイナリー添付ファイルが XOP 文書の前に現れることがあります。 MIME メッセージを処理するアプリケーションは、特定の順序で現れる添付ファイルに依存してはなりません。 詳しくは、MTOM/XOP 仕様を参照してください。

以下の例は、JPEG イメージを含む単純な SOAP メッセージを XOP 処理を使用して最適化する方法を示しています。 SOAP メッセージは次のとおりです。

```
<soap:Envelope
 xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xmime="http://www.w3.org/2003/12/xop/mime">
 <soap:Body>
 <submitClaim>
  <accountNumber>5XJ45-3B2</accountNumber>
  <eventType>accident</eventType>
  <image xmime:contentType="image/jpeg" xsi:type="base64binary">4f3e..(encoded image)</image>
 </submitClaim>
 </soap:Body>
</soap:Envelope>
```

この SOAP メッセージの MTOM/XOP バージョンは以下のとおりです。

```
MIME-Version: 1.0
Content-Type: Multipart/Related; boundary=MIME_boundary;
 type="application/soap+xml"; start="<claim@insurance.com>" 1

--MIME_boundary
Content-Type: application/soap+xml; charset=UTF-8
Content-Transfer-Encoding: 8bit
```

```
Content-ID: <claim@insurance.com> 2

<soap:Envelope
 xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
 xmlns:xop='http://www.w3.org/2004/08/xop/include'
 xmlns:xop-mime='http://www.w3.org/2005/05/xmlmime'>
 <soap:Body>
 <submitClaim>
  <accountNumber>5XJ45-3B2</accountNumber>
  <eventType>accident</eventType>
  <image xop-mime:content-type='image/jpeg'><xop:Include href="cid:image@insurance.com"/></image> 3
 </submitClaim>
 </soap:Body>
</soap:Envelope>

--MIME_boundary
Content-Type: image/jpeg
Content-Transfer-Encoding: binary
Content-ID: <image@insurance.com> 4

...binary JPG image...

--MIME_boundary--
```

1. **start** パラメーターは、MIME メッセージのどの部分がルート XOP 文書であるかを示します。
2. Content-ID 値は MIME メッセージの一部を識別します。 このケースではルート XOP 文書です。
3. <xop:Include> エレメントは JPEG バイナリー添付ファイルを参照します。
4. Content-ID はバイナリー添付ファイル内でこの JPEG を識別します。

CICS における MTOM メッセージとバイナリー添付ファイル

CICS は、MTOM ハンドラー・プログラムと XOP 処理を使用して、Web サービス・プロバイダーと Web サービス・リクエスター両方のパイプラインで、MTOM メッセージの処理をサポートおよび制御します。

MTOM サポートを構成して有効にするには、パイプライン構成ファイルを使用します。 パイプラインで MTOM サポートを有効にすると、CICS によってインバウンド MTOM メッセージが自動的にアンパックされ、アウトバウンド・メッセージがパックされます。 パイプラインで MTOM サポートが有効になっていない状態で CICS が MTOM メッセージを受け取ると、Java ベースのパイプラインは、インバウンド MTOM メッセージを受け入れますが、他の SOAP パイプラインは、インバウンド MTOM メッセージを拒否して、SOAP 障害を生成します。

Java ベースのパイプラインの構成オプション

プロバイダー・パイプラインを構成して以下の作業を実行できます。

- MTOM メッセージを受け入れるが、MTOM 応答メッセージは送信しない。
- MTOM メッセージを受け入れ、常に MTOM 応答メッセージを送信する。
- XOP 文書およびバイナリー添付ファイルを Axis2 モードで処理する。

リクエスター・パイプラインを構成して以下の作業を実行できます。

- MTOM メッセージを送信しないが、MTOM 応答メッセージを受け入れる。
- 常に MTOM メッセージを送信し、MTOM 応答メッセージを受け入れる。
- XOP 文書およびバイナリー添付ファイルを Axis2 モードで処理する。

Java をサポートしないパイプラインの構成オプション

プロバイダー・パイプラインを構成して以下の作業を実行できます。

- MTOM メッセージを受け入れるが、MTOM 応答メッセージは送信しない。
- MTOM メッセージを受け入れ、同じタイプの応答メッセージを送信する。
- MTOM メッセージを受け入れるが、バイナリー添付ファイルが存在する場合にのみ MTOM メッセージを送信する。
- MTOM メッセージを受け入れ、常に MTOM 応答メッセージを送信する。
- XOP 文書およびバイナリー添付ファイルを直接モードまたは互換モードで処理する。

リクエスター・パイプラインを構成して以下の作業を実行できます。

- MTOM メッセージを送信しないが、MTOM 応答メッセージを受け入れる。
- バイナリー添付ファイルがある場合にのみ MTOM メッセージを送信し、MTOM 応答メッセージを受け入れる。
- 常に MTOM メッセージを送信し、MTOM 応答メッセージを受け入れる。
- XOP 文書およびバイナリー添付ファイルを直接モードまたは互換モードで処理する。

サポートの方式

パイプラインでは、XOP 文書と関連バイナリー添付ファイルを処理するための 3 つのモードのサポートが用意されています。

Axis2 モード
: Axis2 モードが使用されるのは、Web サービス・パイプラインの終端ハンドラーが <cics_soap_1.1_handler_java>メッセージ・ハンドラーまたは <cics_soap_1.2_handler_java> メッセージ・ハンドラーの場合です。

直接モード
: 直接モードでは、インバウンドまたはアウトバウンド MTOM メッセージに関連付けられるバイナリー添付ファイルは、パイプラインを通ってコンテナー内に渡され、アプリケーションによって直接処理されます。データ変換の必要はありません。

互換モード
: 互換モードは、パイプライン処理で、メッセージが標準 XML フォーマットであり、メッセージ内に任意のバイナリー・データが base64Binary フィールドとして保管されている必要があるときに、使用されます。 インバウンド・メッセージでは、XOP 文書とバイナリー添付ファイルが標準 XML メッセージに再構成されます。これは、Web Services Security が使用可能になるときのパイプラインの最初か、Web サービスの検証が使用可能になるときのパイプラインの最後に行われます。 アウトバウンド・メッセージでは、標準 XML メッセージはパイプラインに従って作成され、渡されます。 CICS が送信する直前に MTOM ハンドラーによって、この XML メッセージが XOP フォーマットに変換されます。

互換モードでは、バイナリー・データが base64 フォーマットに変換され、再度元に戻るので、直接モードよりかなり効率が悪くなります。 しかし、アプリケーショ

ンを変更せずに、Web サービスとその他の MTOM/XOP Web サービス・リクエスターおよび Web サービス・プロバイダーとを相互運用できます。

Java をサポートしないパイプラインのインバウンド **MTOM** メッセージの処理**:**

Java をサポートしないパイプラインで MTOM ハンドラーが有効になっている場合、DFHREQUEST コンテナーまたは DFHRESPONSE コンテナーでインバウンド・メッセージのヘッダーが検査され、トランスポート処理時のメッセージのフォーマットが決定されます。

MIME Multipart/Related メッセージを受け取ると、MTOM ハンドラーは次のようにこのメッセージをアンパックします。

1. 各バイナリー添付ファイルのヘッダーおよびバイナリー・データを、別のコンテナーに入れる。
2. コンテナーのリストを DFHWS-XOP-IN コンテナーに入れる。
3. メッセージのルートを形成する XOP 文書を、DFHREQUEST または DFHRESPONSE コンテナーに戻し、元のメッセージを置き換える。

バイナリー添付ファイルがない場合は、XOP 文書は通常の XML メッセージとして処理され、 XOP 処理は必要ありません。 バイナリー添付ファイルがある場合は、XOP 処理がメッセージについて使用可能になります。

XOP 処理が使用可能な場合は、MTOM ハンドラーがパイプラインのプロパティーを検査して、現行メッセージを直接モードで処理するか互換モードで処理するかを決定し、この情報を DFHWS-MTOM-IN コンテナーに入れます。

プロバイダー・モードでは、MTOM ハンドラーは DFHWS-MTOM-OUT コンテナーも作成して、アウトバウンド応答メッセージが処理される方法を決定します。

直接モード

CICS Web サービス・サポートを使用している場合 (つまり、サービス・プロバイダー・パイプラインが DFHPITP アプリケーション・ハンドラーを使用するか、サービス・リクエスター・パイプラインが **INVOKE WEBSERVICE** を使用して呼び出される場合)、パイプラインは直接モードで XOP 文書とバイナリー添付ファイルを処理できます。

この方式では、XOP 文書および関連するコンテナーは、MTOM ハンドラーによって、処理のためにパイプライン内の次のメッセージ・ハンドラーに渡されます。 CICS Web サービス・サポートは `<xop:Include>` エレメントを解釈します。 base64Binary フィールドがアプリケーション・データ構造内でコンテナーとして表される場合、添付ファイルのコンテナー名はこの構造に保管されます。 このフィールドが可変または固定長のストリングとして表される場合、コンテナーの内容は該当するアプリケーション・データ構造のフィールドにコピーされます。 それからデータ構造がアプリケーション・プログラムに渡されます。

互換モード

パイプラインがカスタム・アプリケーション・ハンドラーを使用するように構成されている場合、または Web Services Security も使用可能な場合は、メッセージは互換モードで処理されます。 この方式では、XOP 文書とバイナリー添付ファイル

は XOP 処理を使用して即時に SOAP メッセージに再構成されるので、内容はパイプライン内で正常に処理されます。 XOP 処理では以下の作業を行います。

1. <xop:Include> エレメントについて XOP 文書をスキャンして、参照される添付ファイルから、バイナリー・データを持つ各オカレンスを base64 エンコードされたフォーマットに置き換える。
2. DFHWS-XOP-IN コンテナーおよびすべての添付ファイルのコンテナーを破棄する。

再構成された SOAP メッセージは、この後パイプライン内の次のハンドラーに渡され、通常どおりに処理されます。

Web サービスの検証が使用可能であれば、メッセージがアプリケーション・ハンドラーに到達するときに、パイプラインが互換モードに切り替えます。 メッセージは、SOAP メッセージに再構成され、検証されてから、アプリケーションに渡されます。

Java をサポートしないパイプラインのアウトバウンド **MTOM** メッセージの処理**:**

Java をサポートしないパイプラインでアウトバウンド MTOM メッセージを送信するように構成した場合は、Web サービスとパイプラインのプロパティーが検査され、メッセージの処理と送信の方法が決定されます。

これらのプロパティーは、DFHWS-MTOM-OUT と DFHWS-XOP-OUT の 2 つのコンテナーに保管されます。 リクエスター・モードのパイプラインでは、これらのコンテナーは、アプリケーションが **EXEC CICS INVOKE WEBSERVICE** コマンドを出すときに、CICS によって作成されます。 プロバイダー・モードのパイプラインでは、DFHWS-MTOM-OUT コンテナーは、インバウンド・メッセージが受信されたときに決定されたオプションで、すでに初期化されています。

アウトバウンド・メッセージを直接モードで処理できる場合、メッセージの最適化は即時に行われます。 アウトバウンド・メッセージを互換モードで処理する必要がある場合は、最適化はパイプライン処理の最後に行われます。

Web サービス・プロバイダーまたは Web サービス・リクエスターのアプリケーションを CICS Web サービス・アシスタントを使用して配置していない場合、またはパイプラインで Web サービスの検証を使用可能にしているか、Web Services Security を使用可能にしている場合は、アウトバウンド・メッセージは互換モードで処理されます。

直接モード

直接モードでは、次の処理が行われます。

1. XOP 文書は、コンテナー DFHWS-DATA にあるアプリケーションのデータ構造から構成されます。 サイズが 1500 バイト以上のすべてのバイナリー・フィールドが識別され、バイナリー添付ファイルを記述するバイナリー・データおよび MIME ヘッダーが別のコンテナーに入ります。バイナリー・データがすでにコンテナー内にある場合は、そのコンテナーが添付ファイルとして直接使用されます。 次に、生成された Content-ID を使用して、<xop:Include> エレメントが、通常の base64 エンコードされたバイナリー・データの代わりに XML に挿入されます。 以下に例を示します。

```
<xop:Include href="cid:generated-content-ID-value"
 xmlns:xop="http://www.w3.org/2004/08/xop/include">
```

2. すべてのコンテナーが DFHWS-XOP-OUT コンテナー内の添付ファイル・リストに追加されます。
3. SOAP ハンドラーが DFHWS-DATA を処理した場合は、XOP 文書および SOAP エンベロープが DFHREQUEST または DFHRESPONSE コンテナーに保管され、パイプラインを介して処理されます。
4. 最後のメッセージ・ハンドラーが終了すると、MTOM ハンドラーが、XOP 文書およびバイナリー添付ファイルを MIME Multipart/Related メッセージにパックし、Web サービス・リクエスターまたは Web サービス・プロバイダーに送信します。 その後で DFHWS-XOP-OUT コンテナーおよび関連するコンテナーがすべて破棄されます。

互換モード

パイプラインが XOP 文書を直接処理できない場合は、次の処理が行われます。

1. SOAP 本体がアプリケーション・データ構造から DFHWS-DATA に構成され、通常どおりパイプラインで処理されます。
2. 最終ハンドラーがメッセージの処理を終了すると、MTOM ハンドラーが DFHWS-MTOM-OUT コンテナーのオプションを検査し、オプションでバイナリー添付ファイルが存在するかどうかを考慮に入れて、 MTOM を使用するかどうかを決定します。 MTOM ハンドラーが、MTOM は必要ないと判断する場合、 XOP 処理は行われず、SOAP メッセージは CICS によって通常どおりに送信されます。
3. MTOM ハンドラーが、アウトバウンド・メッセージを MTOM フォーマットで送信すると決定すると、データをバイナリー添付ファイルに分割するのに適格なフィールドについて、XOP 処理がメッセージをスキャンします。 適格なフィールドとは、MIME **contentType** 属性がエレメントに指定されていて、関連するバイナリー値が正規形式で有効な base64Binary データからなるものです。データのサイズは 1500 バイトより大きい必要があります。 XOP 処理は、バイナリー添付ファイルと添付リストを作成してから、これらのフィールドを `<xop:Include>` エレメントで置き換えます。
4. MTOM ハンドラーが XOP 文書およびバイナリー添付ファイルを MIME Multipart/Related メッセージとしてパックし、CICS が Web サービス・リクエスターまたは Web サービス・プロバイダーに送信します。

MTOM/XOP 使用の際の制限

MTOM/XOP をサポートするには、パイプライン構成ファイルで `<mtom>` エレメントを指定するか、パイプラインで MTOM ハンドラーを有効にします。 ただし、どちらの方法にも関連した制限があります。

Java ベースのパイプラインの制限**:**

パイプライン構成ファイルで `<mtom>` エレメントを指定すると、Java ベースのパイプラインで MTOM/XOP サポートが有効になります。 ただし、この MTOM/XOP 実装については、いくつかの制限があります。

DFHPITP アプリケーション・ハンドラー
: アプリケーション・ハンドラーとして DFHPITP を指定したパイプラインでは、Axis2 モードの MTOM/XOP サポートを使用できません。

WS-Security
: XML 署名が必要な WS-Security 構成を使用するパイプラインでは、Axis2 モードの MTOM/XOP サポートを使用できません。

INQUIRE PIPELINE コマンドの使用
: Axis2 モードの MTOM/XOP サポートを使用する Java ベースのパイプラインに対して **INQUIRE PIPELINE** コマンドを実行すると、**Mtomst**、**Sendmtomst**、**Mtomnoxopst**、**Xopsupportst**、**Xopdirectst** の各属性が Nomtom として報告されます。詳しくは、INQUIRE PIPELINE を参照してください。

他の **SOAP** パイプラインの制限**:**

パイプラインで MTOM ハンドラーを使用可能にすると、MTOM/XOP 最適化を使用する Web サービスの実装をサポートできます。互換モードのオプションでは、Web サービス・アプリケーションを変更せずに、Web サービスと相互運用できます。ただし、ある状態では、MTOM/XOP を使用できないか、使用が制限されます。

CICS Web サービス・アシスタントの使用
: MTOM/XOP の直接モード最適化を使用できるのは、少なくともマッピング・レベル 1.2 の DFHWS2LS を使用し、xsd:base64Binary タイプのフィールドが WSDL 文書に少なくとも 1 つ含まれる場合だけです。
DFHLS2WS を使って有効化された Web サービスは XOP 最適化を使用できません。

  CHAR-VARYING=BINARY が指定された DFHLS2WS によって生成される Web サービスは、MTOM/XOP 最適化を使用できる可能性があります。 DFHLS2WS を使って生成された他の Web サービスにはバイナリー・データが含まれず、MTOM/XOP 最適化を使用することはできませんが、MTOM/XOP をサポートする PIPELINE では正常に作動します。

プロバイダー・パイプライン
: CICS は、プロバイダー・パイプラインで構成できる、DFHPITP と呼ばれるデフォルトのアプリケーション・ハンドラーを提供します。 このアプリケーション・ハンドラーでは、XOP 文書を処理し、必要なコンテナーを作成して、直接モードと互換モードの両方でのパイプライン処理をサポートできます。 プロバイダー・パイプラインで独自のアプリケーション・ハンドラーを使用していて、 MTOM/XOP を使用可能に設定したい場合は、パイプラインを構成して互換モードで実行します。

リクエスター・パイプライン
: アプリケーションで **INVOKE WEBSERVICE** コマンドを使用する場合、 CICS は SOAP メッセージの最適化を直接モードおよび互換モードで処理します。 プログラム DFHPIRT を使用してパイプラインを開始する場合、互換モードで MIME Multipart/Related メッセージの送受信のみ行えます。

Web Services Security
: パイプライン構成ファイルで MTOM ハンドラーを使用可能にして直接モ

ードで実行する場合で、Web Services Security メッセージ・ハンドラーも使用可能にする場合、パイプラインは、互換モードで MTOM メッセージの処理のみサポートします。

バイナリー・データの処理
: Web サービスに大容量のバイナリー・データ (例えば JPEG などのグラフィック・ファイル) を組み込む場合、MTOM/XOP を使用して、サービス・プロバイダーまたはサービス・リクエスターに送信されるメッセージのサイズを最適化できます。 MTOM/XOP を使用して最適化できるバイナリー・データの最小サイズは 1500 バイトです。 フィールド内のバイナリー・データが 1500 バイトより小さいと、CICS はそのフィールドを最適化しません。

  XOP 仕様に記載されるように、base64Binary データに空白文字を入れることはできません。 base64Binary データを作成するすべてのアプリケーション・プログラムで正規形式を使用する必要があります。 アウトバウンド・メッセージの base64Binary データに空白文字が含まれていると、CICS はそのデータをバイナリー添付ファイルに変換しません。 base64Binary データが CICS によって生成される場合は、フィールドが正規形式で提供されるため、空白文字は含まれません。

  アウトバウンド・メッセージで、互換モードで XOP 処理を行うには、**contentType** 属性が base64Binary フィールドに存在しなければなりません。 **contentType** 属性が hexBinary フィールドに存在していてはなりません。

Web サービスの検証
: Web サービスの検証を有効にすると、次のパイプライン処理が実行されます。

  - インバウンド XOP 文書が直接モードでパイプラインを通過すると、CICS は自動的に互換モードに切り替わり、CICS Web サービス・サポートが文書を検証しようとするときに、元の標準 XML に変換します。
  - アウトバウンド SOAP メッセージは標準 XML として生成され、互換モードで処理されます。

  検証処理で XOP 文書の内容を処理できないため、追加のパイプライン処理が必要です。

MTOM/XOP をサポートするための CICS の構成

CICS で MTOM メッセージをサポートするには、パイプラインのタイプに対応する正しい MTOM/XOP サポートをパイプライン構成ファイルで指定する必要があります。

Java ベースのパイプラインの **MTOM/XOP** サポートの構成**:**

Java ベースのパイプラインの MTOM/XOP サポートを構成するには、パイプライン構成ファイルに `<mtom>` エレメントを追加する必要があります。

始める前に

この作業を行う前に、MTOM/XOP の構成情報を追加するパイプライン構成ファイルを、識別するか、作成する必要があります。

このタスクについて

<mtom> エレメントをパイプライン構成ファイルで定義すると、すべてのインバウンド・メッセージとアウトバウンド・メッセージで MTOM サポートが有効になります。 一方、このエレメントをパイプライン構成ファイルで指定しない場合は、インバウンド・メッセージだけで MTOM サポートが有効になります。

手順

パイプライン構成ファイルに <mtom> エレメントを追加します。 このエレメントは、オプションの <addressing> エレメントとオプションの <headerprogram> エレメントの間に定義する必要があります。

例

プロバイダー・モードまたはリクエスター・モードのパイプラインで、以下のように指定できます。

```
<cics_soap_1.2_handler_java>
  <jvmserver>JVMSERV1</jvmserver>
  <addressing></addressing>
  <mtom></mtom>
  <headerprogram>
    <program_name>HDRPROG4</program_name>
    <namespace>http://mynamespace</namespace>
    <localname>myheaderblock</localname>
    <mandatory>true</mandatory>
  </headerprogram>
</cics_soap_1.2_handler_java>
```

他の **SOAP** パイプラインの **MTOM/XOP** の構成**:**

<cics_soap_1.1_handler_java> ハンドラーまたは <cics_soap_1.2_handler_java> ハンドラーを使用しないパイプラインの MTOM/XOP サポートを構成するには、パイプライン構成ファイルに MTOM ハンドラーを追加する必要があります。

始める前に

この作業を行う前に、MTOM/XOP の構成情報を追加するパイプライン構成ファイルを、識別するか、作成する必要があります。

手順

1. パイプライン構成ファイルに <cics_mtom_handler> エレメントを追加する。 このエレメントが、<provider_pipeline> エレメントの最初になり、<requester_pipeline> エレメント内の <service_parameter_list> の直前のエレメントになります。 次のエレメントをコーディングします。

   ```
   <cics_mtom_handler>
     <dfhmtom_configuration version="1">
     </dfhmtom_configuration>
   </cics_mtom_handler>
   ```

   <dfhmtom_configuration> エレメントは、この構成内の他のエレメントのコンテナーです。 MTOM/XOP 処理のデフォルト設定を受け入れる場合は、以下のように空のエレメントを指定できます。

   ```
     <cics_mtom_handler/>
   ```

2. オプション: <mtom_options> エレメントをコーディングする。 このエレメントは、サービス・プロバイダーおよびサービス・リクエスターの両方のパイプラインで、アウトバウンド・メッセージを MTOM メッセージとしてパックするかどうかを指定します。

   a. **send_mtom** 属性をコーディングして、アウトバウンド・メッセージが MTOM メッセージとして送信されるかどうかを定義する。 この属性について詳しくは、 140 ページの『<mtom_options> パイプライン構成エレメント』を参照してください。

   b. **send_when_no_xop** 属性をコーディングして、バイナリー添付ファイルが存在しないときに、アウトバウンド・メッセージが MTOM メッセージとして送信されるかどうかを定義する。 この属性について詳しくは、 140 ページの『<mtom_options> パイプライン構成エレメント』を参照してください。

3. オプション: <xop_options> エレメントを **apphandler_supports_xop** 属性と一緒にコーディングする。 これで、アプリケーション・ハンドラーが XOP 文書を直接処理できるかどうかを指定します。この属性を組み込まない場合、デフォルトは、<apphandler> エレメントが DFHPITP を指定するか別のプログラムを指定するかに応じて異なります。この属性について詳しくは、 142 ページの『<xop_options> パイプライン構成エレメント』を参照してください。

4. オプション: <mime_options> エレメントを **content_id_domain** 属性と一緒にコーディングする。 これで、バイナリー添付ファイルの識別に使用される MIME content-ID 値を生成する際に使用する、ドメイン・ネームを指定します。 この属性について詳しくは、 144 ページの『<mime_options> パイプライン構成エレメント』を参照してください。

例

次の例は、オプションのエレメントがすべて存在する、完全な <cics_mtom_handler> エレメントを示します。

```
<provider_pipeline>
  <cics_mtom_handler>
    <dfhmtom_configuration version="1">
      <mtom_options send_mtom="same" send_when_no_xop="no" />
      <xop_options apphandler_supports_xop="yes" />
      <mime_options content_id_domain="example.org" />
    </dfhmtom_configuration>
  </cics_mtom_handler>
....
</provider_pipeline>
```

Web Services Addressing のサポート

CICS は、Worldwide Web Consortium (W3C) の Web Services Addressing (WS-Addressing) 仕様を使用するサービスをサポートします。 この仕様ファミリーは、Web サービスのアドレッシングとエンドツーエンドのアドレッシングを可能にする、トランスポートに依存しない機構を提供します。

CICS は、WS-Addressing を使用する Web サービスからの要求を既存の Web サービス・アプリケーションで受け入ることができることを保証します。 SOAP メッセージのエンドポイント参照とメッセージ・アドレッシング・プロパティーを使用する Web サービスを新規作成することもできます。

WS-Addressing は、メッセージ・アドレッシング・プロパティー (MAP) の形式のアドレッシング情報を SOAP メッセージ・ヘッダーに追加します。 MAP には、固有のメッセージ ID や、エンドポイント参照 (メッセージの送信元、メッセージの送信先、および応答メッセージや障害メッセージの送信先が詳しく説明されている) などのメッセージング情報が含まれます。 エンドポイント参照 (EPR) は特定のタイプの MAP です。これには、メッセージの宛先アドレス、アプリケーションが使用するオプションの参照パラメーター、およびオプションのメタデータが含まれます。

WS-Addressing サポートの機能

CICS には、WS-Addressing をサポートする以下のフィーチャーが含まれます。

- Web サービス・リクエスター・アプリケーションと Web サービス・プロバイダー・アプリケーションは、再デプロイを行わずに、WS-Addressing を使用しているその他のサービスと対話することができます。 パイプラインにある新しいメッセージ・ハンドラー、すなわちアドレッシング・メッセージ・ハンドラー DFHWSADH は、WS-Addressing 情報を含むメッセージを指定の Web サービスに経路指定します。
- WS-Addressing API コマンドを使用してエンドポイント参照を作成し、アドレッシング・コンテキストを作成、更新、削除、および照会するアプリケーションを作成することができます。
- 応答メッセージをリクエスター・エンドポイント以外のエンドポイントに経路指定することができます。例えば、障害メッセージを専用の障害ハンドラーに経路指定することができます。
- 参照パラメーターを SOAP ヘッダー内の MAP の一部としてアプリケーションに渡すことができます。

WS-Addressing 仕様のサポートと相互運用性

デフォルトで、CICS は以下の勧告仕様をサポートしています。

- W3C WS-Addressing 1.0 - Core
- W3C WS-Addressing 1.0 - SOAP Binding
- W3C WS-Addressing 1.0 - Metadata

これらの仕様は、`http://www.w3.org/2005/08/addressing` の名前空間によって識別されます。特に明記しない限り、本書に記載される WS-Addressing セマンティクスは勧告仕様のことです。

相互運用性に関しては、CICS は実行依頼仕様もサポートしています。

- W3C WS-Addressing- Submission

この仕様は、`http://schemas.xmlsoap.org/ws/2004/08/addressing` の名前空間によって識別されます。処理依頼仕様は、それをインプリメントするクライアントまたは Web サービス・プロバイダーと相互運用する必要がある場合のみ使用します。

Web Services Addressing の概要

Web Services Addressing (WS-Addressing) は、SOAP メッセージのエンドポイントを指定するための標準的なフレームワークです。 このフレームワークはトランスポートに中立的であるため、さまざまなトランスポート機構を使用する Web サー

ビスの相互運用性が改善されます。 WS-Addressing 仕様は、メッセージ・アドレッシング・プロパティーおよびエンドポイント参照を導入しています。

Worldwide Web Consortium (W3C) の仕様の 1 つである Web Services Addressing (WS-Addressing) は、Web サービスをアドレス指定する標準的な方法を定義し、SOAP メッセージ内のアドレッシング情報を提供することにより、Web サービス間の相互運用性を改善します。 HTTP や WebSphere MQ などのさまざまなトランスポート機構を介して SOAP メッセージを送ることができます。それぞれの機構は、他とは異なる方法でメッセージの宛先情報を保管します。

WS-Addressing を使用するように構成されたパイプラインにデプロイされている既存の CICS Web サービスでは、デフォルトの WS-Addressing 設定を変更せずに、そのまま使用できます。 WS-Addressing 機能を十分に利用するためには、WS-Addressing API コマンドを使用します。WS-Addressing 実装は、WSDL 操作ごとに 1 つの SOAP 障害をサポートします。

メッセージ・アドレッシング・プロパティー

メッセージ・アドレッシング・プロパティー (MAP) は、SOAP ヘッダー内のエレメントとして表すことができる、明確に定義された WS-Addressing プロパティーから成るセットです。 MAP は、メッセージ応答の送信先となるエンドポイントや、そのメッセージと他のメッセージの関係を示す情報などを伝達する標準的な方法として機能します。 以下の表は、WS-Addressing 仕様で定義されている MAP の要約です。

表 *11. WS-Addressing* 仕様で定義されているメッセージ・アドレッシング・プロパティー

抽象 **WS-Addressing MAP** 名	**SOAP WS-Addressing MAP** 名	**MAP** コンテンツ・タイプ	多重度	説明
[action]	<wsa:Action>	xs:anyURI	1..1	メッセージのセマンティクスを一意的に識別する絶対 URI。 この値は必須です。
[destination]	<wsa:To>	SOAP メッセージ内の xs:anyURI アドレッシング・コンテキストでの EndpointReference	0..1	期待されるメッセージ受信側のアドレスを指定する絶対 URI。 この値が指定されない場合、デフォルトの、仕様で定義された匿名 URI `http://www.w3.org/2005/08/addressing/anonymous` になります。 アドレッシング・コンテキストでは、<wsa:To> MAP は EPR として表されます。 <wsa:To> が SOAP メッセージの一部として送信されると、それはスキーマで定義されているようにアドレスおよびその参照パラメーターに分割されます。
[reference parameters] *	[reference parameters]*	xs:any	0..制限なし	メッセージがアドレス指定されるエンドポイント参照の <wsa:ReferenceParameters> プロパティーに対応するパラメーター。この値はオプションです。
[source endpoint]	<wsa:From>	EndpointReference	0..1	メッセージの発信元となったエンドポイントの参照。 この値はオプションです。

表 *11. WS-Addressing* 仕様で定義されているメッセージ・アドレッシング・プロパティー *(*続き*)*

抽象 **WS-Addressing MAP** 名	**SOAP WS-Addressing MAP** 名	**MAP** コンテンツ・タイプ	多重度	説明
[reply endpoint]	<wsa:ReplyTo>	EndpointReference	0..1	このメッセージの応答を受け取ることが期待されるエンドポイントの参照。 この値はオプションです。 この値が指定されない場合、デフォルトとして `http://www.w3.org/2005/08/addressing/anonymous` が使用されます。
[fault endpoint]	<wsa:FaultTo>	EndpointReference	0..1	このメッセージに関連した障害を受け取ることが期待されるエンドポイントの参照。 この値はオプションで、デフォルトは <wsa:ReplyTo> MAP の値となります。
[relationship] *	<wsa:RelatesTo>	xs:anyURI に加えて、タイプ xs:anyURI のオプション属性	0..制限なし	このメッセージと別のメッセージの関係を示す値のペア。 このエレメントの内容は、関連するメッセージの <wsa:MessageID> を伝送します。 オプション属性は関係のタイプを伝送します。 この値はオプションです。 この値が指定されない場合、デフォルトとして `http://www.w3.org/2005/08/addressing/reply` が使用されます。
[message id]	<wsa:MessageID>	xs:anyURI		メッセージを一意的に識別する絶対 URI。 この値はオプションで、提供されない場合は CICS がアウトバウンド要求および応答の値を生成します。

以下のサンプル SOAP メッセージには WS-Addressing MAP が含まれています。

```
<S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope"
            xmlns:wsa="http://w3.org/2005/08/addressing"
            xmlns:example="http://example.ibm.com/namespace">
  <S:Header>
      ...
      <wsa:To>http://example.ibm.com/enquiry</wsa:To>
      <wsa:ReplyTo>
        <wsa:Address>http://example.ibm.com/enquiryReply</wsa:Address>
      </wsa:ReplyTo>
      <wsa:Action>...</wsa:Action>
      <example:AccountCode wsa:IsReferenceParameter='true'>123456789</example:AccountCode>
      <example:DiscountId wsa:IsReferenceParameter='true'>ABCDEFG</example:DiscountId>
      ...
  </S:Header>
  <S:Body>
  ...
  </S:Body>
</S:Envelope>
```

エンドポイント参照

エンドポイント参照は、特定のエンドポイントについての情報をカプセル化する標準的な手段を提供する、特定のタイプの MAP です。 エンドポイント参照を他のパーティーに送信して、それらが表す Web サービス・エンドポイントをターゲットにするために使用できます。 以下の表は、エンドポイント参照の情報モデルの要約です。

表 12. エンドポイント参照の情報モデル

抽象プロパティー名	プロパティー・タイプ	多重度	説明
[address]	xs:anyURI	1..1	エンドポイントのアドレスを指定する絶対 URI。
[reference parameters] *	xs:any	0..制限なし	エンドポイントとインターフェースを取るために必要な、名前空間で修飾されたエレメント情報項目。
[metadata]	xs:any	0..制限なし	エンドポイントの動作、方針、および機能に関する説明。

次の XML 断片は、エンドポイント参照を例示しています。 `<wsa:EndpointReference>` エレメントは、 URI `http://example.ibm.com/enquiry` でエンドポイントを参照し、エンドポイント参照先のインターフェースを指定するメタデータおよびアプリケーション固有のいくつかの参照パラメーターを含んでいます。

```
<wsa:EndpointReference
    xmlns:wsa="http://www.w3.org/2005/08/addressing"
    xmlns:wsam="http://www.w3.org/2007/05/addressing/metadata"
    xmlns:example="http://example.ibm.com/namespace">
  <wsa:Address>http://example.ibm.com/enquiry</wsa:Address>
  <wsa:Metadata
      xmlns:wsdli="http://www.w3.org/ns/wsdl-instance"
      wsdli:wsdlLocation="http://example.ibm.com/wsdl/wsdl-location.wsdl">
    <wsam:InterfaceName>example:reservationInterface</wsam:InterfaceName>
  </wsa:Metadata>
  <wsa:ReferenceParameters>
      <example:AccountCode>123456789</example:AccountCode>
      <example:DiscountId>ABCDEFG</example:DiscountId>
   <wsa:ReferenceParameters>
</wsa:EndpointReference>
```

タイプ `wsa:EndpointReferenceType` の WS-Addressing MAP は、<wsa:From>、<wsa:ReplyTo>、および <wsa:FaultTo> です。 ただし <wsa:To> MAP は、WS-Addressing 1.0 標準ではタイプ `xs:anyURI` として定義されています。分かりやすくするために、CICS はアドレッシング・コンテキストでは <wsa:To> MAP を EPR として扱います。 <wsa:To> MAP が SOAP メッセージの一部として送信されると、 CICS は標準の必要に応じて、そのアドレスおよび参照パラメーターに分割します。

デフォルトの名前空間

以下の接頭部およびそれに対応する名前空間は、WS-Addressing の文書全体で参照されます。

表 13. 接頭部および対応する名前空間

デフォルトの接頭部	名前空間
xs	http://www.w3.org/2001/XMLSchema
wsa	http://www.w3.org/2005/08/addressing (勧告スキーマ) http://schemas.xmlsoap.org/ws/2004/08/addressing (サブミッション・スキーマ)
wsam	http://www.w3.org/2007/05/addressing/metadata

Web Services Addressing に合わせたリクエスター・パイプラインの構成

Web Services Addressing (WS-Addressing) をサポートするようにリクエスター・パイプラインを構成するには、パイプライン構成ファイルにアドレッシング・ハンドラーを追加する必要があります。

始める前に

WS-Addressing の構成情報を追加するパイプライン構成ファイルを識別するか、または作成する必要があります。 どの WS-Addressing 仕様を使用するかも決定する必要があります。 可能であれば、*W3C WS-Addressing 1.0 Core* 仕様を使用してください。

このタスクについて

次の 2 つの方法のいずれかで、WS-Addressing のサポートを追加できます。

- SOAP パイプラインで Java を使用する場合、Axis2 によって SOAP 処理が扱われ、このテクノロジーで提供されているサポートを使用して、WS-Addressing を使用する要求を処理できます。 ヘッダー処理はすべて Axis2 によって扱われます。DFHWSADH ヘッダー処理プログラムはパイプラインに追加しないでください。 独自のヘッダー処理プログラムを使用することは可能です。 SOAP ヘッダーを処理する場合、Axis2 ハンドラーを Java で作成すると、パフォーマンスが向上します。
- SOAP パイプラインで Java を使用しない場合、要求を扱うために、CICS 提供のヘッダー処理プログラム DFHWSADH を追加する必要があります。

手順

- SOAP パイプラインで `<cics_soap_1.1_handler_java>` または `<cics_soap_1.2_handler_java>` エレメントを使用する場合、`<addressing>` エレメントをパイプライン構成ファイルに追加します。 要求メッセージで使用する仕様を入れる `<namespace>` エレメントを 1 つ含めます。これは、応答メッセージとは別のものにできます。例えば、応答メッセージで実行依頼仕様を使用する場合でも、W3C core 仕様に準拠する要求を常に送信できます。 Axis2 は、インバウンド・メッセージで両方の WS-Addressing 仕様をサポートします。

  以下の例は、リクエスター・パイプラインを構成する方法を示しています。

```
<requester_pipeline>
   <service>
      <service_handler_list>
         <cics_soap_1.1_handler_java>
            <jvmserver>JVMSERV1</jvmserver>
            <addressing>
               <namespace>http://www.w3.org/2005/08/addressing</namespace>
            </addressing>
         </cics_soap_1.1_handler_java>
      </service_handler_list>
   </service>
</requester_pipeline>
```

<jvmserver> エレメントには、Axis2 をサポートする JVMSERVER リソースの名前が入ります。

- SOAP パイプラインで Java を使用しない場合、<cics_soap_1.1_handler> または <cics_soap_1.2_handler> の CICS アドレス指定ヘッダー・プログラムをパイプライン構成ファイルに追加します。 以下の例は、リクエスター・パイプラインを構成する方法を示しています。

```
<requester_pipeline>
   <service>
      <service_handler_list>
         <cics_soap_1.1_handler>
            <headerprogram>
               <program_name>DFHWSADH</program_name>
               <namespace>http://www.w3.org/2005/08/addressing</namespace>
               <localname>*</localname>
               <mandatory>true</mandatory>
            </headerprogram>
         </cics_soap_1.1_handler>
      </service_handler_list>
   </service>
</requester_pipeline>
```

  ここに示されているとおりに、<program_name>、<localname>、および <mandatory> エレメントを正確にコーディングします。 <namespace> は、*W3C WS-Addressing 1.0 Core* 仕様を使用する場合には http://www.w3.org/2005/08/addressing に設定し、*W3C WS-Addressing Submission* 仕様を使用する場合には http://schemas.xmlsoap.org/ws/2004/08/addressing に設定します。

  ヘッダー処理プログラムの順序は保障されません。他のヘッダー処理プログラムを定義する場合、それらを <service_handler_list> エレメント内の以降の CICS SOAP ハンドラー・エレメントに追加します。 DFHWSADH ヘッダー・ハンドラーは、最初の SOAP ハンドラー・エレメントになければなりません。

タスクの結果

これで、WS-Addressing をサポートするように、リクエスター・パイプラインが構成されました。

次のタスク

構成ファイルを指す PIPELINE リソースを作成します。 Java ベースの SOAP パイプラインを使用する場合、Axis2 処理を扱うために、JVMSERVER リソースが有効になっていることを確認します。

Web Services Addressing に合わせたプロバイダー・パイプラインの構成

Web Services Addressing (WS-Addressing) をサポートするようにプロバイダー・パイプラインを構成するには、パイプライン構成ファイルにアドレッシング・ハンドラーを追加する必要があります。

始める前に

WS-Addressing の構成情報を追加するパイプライン構成ファイルを識別するか、または作成する必要があります。 どの WS-Addressing 仕様を使用するかも決定する

必要があります。 可能であれば、*W3C WS-Addressing 1.0 Core* 仕様を使用してください。

このタスクについて

次の 2 つの方法のいずれかで、WS-Addressing のサポートを追加できます。

- SOAP パイプラインで Java を使用する場合、Axis2 によって SOAP 処理が扱われ、このテクノロジーで提供されているサポートを使用して、WS-Addressing を使用する要求を処理できます。 ヘッダー処理はすべて Axis2 によって扱われます。DFHWSADH ヘッダー処理プログラムはパイプラインに追加しないでください。 独自のヘッダー処理プログラムを使用することは可能です。 SOAP ヘッダーを処理する場合、Axis2 ハンドラーを Java で作成すると、パフォーマンスが向上します。
- SOAP パイプラインで Java を使用しない場合、要求を扱うために、CICS 提供のヘッダー処理プログラム DFHWSADH を追加する必要があります。

手順

- SOAP パイプラインで <cics_soap_1.1_handler_java> または <cics_soap_1.2_handler_java> エレメントを使用する場合、<addressing> エレメントをパイプライン構成ファイルに追加します。 1 つ以上の <namespace> エレメントをオプションで含めることができます。このエレメントには、アウトバウンド・メッセージで使用する仕様を含めます。これは、インバウンド・メッセージとは別のものにできます。例えば、インバウンド・メッセージで実行依頼仕様を使用する場合でも、W3C core 仕様に準拠するアウトバウンド応答を常に送信できます。 このエレメントを除外すると、Axis2 はインバウンド・メッセージと同じ仕様をアウトバウンド・メッセージで使用します。 Axis2 は、インバウンド・メッセージで両方の WS-Addressing 仕様をサポートします。

  以下の例は、プロバイダー・パイプラインを構成する方法を示しています。

```
<provider_pipeline>
   <terminal_handler>
      <cics_soap_1.1_handler_java>
          <jvmserver>JVMSERV1</jvmserver>
          <addressing>
             <namespace>http://www.w3.org/2005/08/addressing</namespace>
          </addressing>
      </cics_soap_1.1_handler_java>
    </terminal_handler>
</provider_pipeline>
```

  <jvmserver> エレメントには、Axis2 をサポートする JVMSERVER リソースの名前が入ります。

- SOAP パイプラインで Java を使用しない場合、CICS アドレス指定ヘッダー・プログラム DFHWSADH をパイプライン構成ファイル内の SOAP ハンドラーに追加します。 以下の例は、プロバイダー・パイプラインを構成する方法を示しています。

```
<provider_pipeline>
   <terminal_handler>
      <cics_soap_1.1_handler>
         <headerprogram>
             <program_name>DFHWSADH</program_name>
             <namespace>http://www.w3.org/2005/08/addressing</namespace>
             <localname>*</localname>
```

```
            <mandatory>true</mandatory>
          </headerprogram>
        </cics_soap_1.1_handler>
    </terminal_handler>
</provider_pipeline>
```

ここに示されているとおりに、<program_name>、<localname>、および <mandatory> エレメントを正確にコーディングします。 <namespace> は、*W3C WS-Addressing 1.0 Core* 仕様を使用する場合には http://www.w3.org/2005/08/addressing に設定し、*W3C WS-Addressing Submission* 仕様を使用する場合には http://schemas.xmlsoap.org/ws/2004/08/addressing に設定します。

ヘッダー処理プログラムの順序は保障されません。他のヘッダー処理プログラムを定義する場合、それらを <service_handler_list> エレメント内の別の CICS SOAP ハンドラー・エレメントに追加します。 DFHWSADH ヘッダー・ハンドラーは、最後の SOAP ハンドラー・エレメントになければなりません。

タスクの結果

これで、WS-Addressing をサポートするように、プロバイダー・パイプラインが構成されました。

次のタスク

構成ファイルを指す PIPELINE リソースを作成します。 Java ベースの SOAP パイプラインを使用する場合、Axis2 処理を扱うために、JVMSERVER リソースが有効になっていることを確認します。

WS-Addressing を使用する Web サービスの作成

Web Services Addressing (WS-Addressing) を使用する WSDL 文書から Web サービスを作成するには、Web サービス・アシスタントで、XML から言語構造への変換を扱うパラメーターを使用します。

このタスクについて

Web サービス・アシスタント・ジョブ DFHWS2LS を使用すると、WSDL 文書でエンドポイント参照 (EPR) を扱う方法の制御と、CICS がデフォルトの入力アクション、出力アクション、および障害アクションを作成するかどうかの判別を行うことができます。

手順

1. Web サービス・アシスタント DFHWS2LS の **MINIMUM-RUNTIME** パラメーターを 3.0 以上に設定します。 3.0 以上のランタイム・レベルでは、生成される Web サービス・バインディングが WS-Addressing を完全にサポートし、他の Web サービス・プラットフォームとも確実に相互運用できます。
2. Web サービス・アシスタント DFHWS2LS の **MAPPING-LEVEL** パラメーターを 3.0 以上に設定します。
3. 要求メッセージまたは応答メッセージで wsa:EndpointReferenceType タイプ・エレメントを使用する場合、**WSADDR-EPR-ANY** パラメーターを TRUE に設定します。 エンドポイント参照はアプリケーション・データに含めることが可能で、**WSACONTEXT BUILD** などの API コマンドで EPR を使用することもできます。

WSADDR-EPR-ANY パラメーターを TRUE に設定する場合、 CICS は EPR を言語構造に実行時に変換することはしません。代わりに CICS は、EPR データを <xsd:any> エレメントとして扱い、指定されたコンテナーに格納します。

以下の例の WSDL フラグメントは、タイプ wsa:EndpointReferenceType のエレメントとして渡される <wsa:To> MAP を示しています。

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="exampleEPR" targetNamespace="http://example.ibm.com/"
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:s0="http://example.ibm.com/"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:wsa="http://www.w3.org/2005/08/addressing"
    xmlns:wsam="http://www.w3.org/2007/05/addressing/metadata">
  <types>
    <xs:schema targetNamespace="http://test.org/"
            xmlns:s="http://www.w3.org/2001/XMLSchema"
            xmlns:s0="http://example.ibm.com/"
            xmlns:wsa="http://www.w3.org/2005/08/addressing">
      ...
       <xs:element name="exampleResponse" type="s0:typeResponse"/>
       <xs:complexType name="typeResponse">
         <xs:sequence>
           <xs:element name="myEpr" type="wsa:EndpointReferenceType"/> 1
         </xs:sequence>
       </xs:complexType>
       ...
    </xs:schema>
  </types>
  ...
  <message name="msgResponse">
    <part element="s0:exampleResponse" name="response"/>
  </message>
  ...
</definitions>
```

WSADDR-EPR-ANY パラメーターを TRUE に設定した DFHWS2LS によってエレメント <xs:element name="myEpr" type="wsa:EndpointReferenceType"/> 1 が処理されると、 myEpr エレメント・データは指定されたコンテナーに <xsd:any> エレメントとして格納され、生成される言語構造にそのコンテナーへのポインターとして追加されます。

例えば、myEpr エレメントに関して DFHWS2LS によって生成される COBOL 言語構造をここに示します。

```
09 myEpr.
   12 myEpr-xml-cont               PIC X(16).
   12 myEpr-xmlns-cont             PIC X(16).
```

myEpr-xml-cont コンテナーは、myEpr データを格納するコンテナーの名前を格納します。 myEpr-xmlns-cont は、スコープ内にある XML 名前空間宣言を取り込むオプションのコンテナーです。

4. DFHWS2LS ジョブを保管し、実行依頼します。

タスクの結果

サービス・リクエスター・アプリケーションまたはサービス・プロバイダー・アプリケーションを作成するために使用できる、データ変換および言語構造を処理するための Web サービス・バインディングが、CICS によって作成されます。

次のタスク

Web サービスを使用可能にするには、パイプライン・スキャンを実行して、必要な CICS リソースを作成します。

デフォルトのエンドポイント参照:

ほとんどの WSDL 文書には、Web サービスがホストされるアドレスが含まれています。 WS-Addressing では、WSDL 文書に Web サービスのエンドポイント参照 (EPR) を含めることもできます。 この EPR には、リクエスター・アプリケーションとプロバイダー・アプリケーションの間での通信を容易にする、追加メタデータが含まれます。

DFHWS2LS を使用して WSDL を処理する場合、EPR は Web サービス・バインディングに保管され、要求および応答メッセージを送信するために CICS によって使用されます。 EPR で定義される参照パラメーター <wsa:ReferenceParameters> は、SOAP メッセージに含められます。 この EPR はアプリケーションによって指定変更される場合があるため、デフォルト EPR と呼ばれます。 アプリケーションで明示的な EPR が指定されない場合、WSDL からのデフォルト EPR が使用されます。

以下の WSDL 1.1 フラグメントには、デフォルト EPR `<soap:address location="http://example.ibm.com:12345/exampleTest" />` が含まれています。 `<port>` エレメントには子エレメント `<wsa:EndpointReference>` が含まれており、子エレメントによって指定されるアドレス **2** は親エレメントによって指定されるアドレス **1** と一致する必要があります。

```
<service name="exampleService">
   <port name="examplePort" binding="s0:createBinding">
      <soap:address location="http://example.ibm.com:12345/exampleTest" /> 1

      <wsa:EndpointReference
             xmlns:example="http://example.ibm.com/namespace"
             xmlns:wsdli="http://www.w3.org/2006/01/wsdl-instance"
             wsdli:wsdlLocation="http://example.ibm.com/location "
             title="http://example.ibm.com/example/example_wsdl"
             class="http://example.ibm.com/example/example.wsdl">
             <wsa:Address>http://example.ibm.com:12345/exampleTest</wsa:Address> 2
             <wsa:Metadata>
                <wsam:InterfaceName>example:Inventory</wsam:InterfaceName>
             </wsa:Metadata>
             <wsa:ReferenceParameters>
               <example:AccountCode>123456789</example:AccountCode>
               <example:DiscountId>ABCDEFG</example:DiscountId>
             </wsa:ReferenceParameters>
      </wsa:EndpointReference>
   </port>
</service>
```

明示的アクション:

WSDL 文書では <wsa:Action> プロパティーの値を明示的に定義できます。 明示的に定義された <wsa:Action> プロパティーが WSDL 文書に含まれていない場合、WSDL が DFHWS2LS によって処理されるときに、CICS はデフォルトのアクションを作成します。

WSDL 1.1

以下の WSDL 1.1 フラグメントは、明示的に定義された <wsa:Action> プロパティーが含まれる予約システムを表しています。

```
<definitions targetNamespace="http://example.ibm.com/namespace" ...>
   ...
   <portType name="bookingSystem">
      <operation name="makeBooking">
         <input message="tns:makeBooking"
            wsa:Action="http://example.ibm.com/namespace/makeBooking"
         </input>
         <output message="tns:bookingResponse"
             wsa:Action="http://example.ibm.com/namespace/bookingResponse"
         </output>
      </operation>
   </portType>
   ...
</definitions>
```

この例で、makeBooking 操作の入力アクションは明示的に http://example.ibm.com/namespace/makeBooking に定義され、出力アクションは明示的に http://example.ibm.com/namespace/bookingResponse に定義されています。

WSDL 2.0

以下の WSDL 2.0 フラグメントは、明示的に定義された <wsa:Action> プロパティーが含まれる予約システムを表しています。

```
<description targetNamespace="http://example.ibm.com/namespace" ...>
  ...
  <interface name="bookingInterface">
    <operation name="makeBooking" pattern="http://www.w3.org/ns/wsdl/in-out">
      <input element="tns:makeBooking" messageLabel="In"
            wsa:Action="http://example.ibm.com/namespace/makeBooking"/>
      <output element="tns:makeBookingResponse" messageLabel="Out"
            wsa:Action="http://example.ibm.com/namespace/makeBookingResponse"/>
    </operation>
  </interface>
  ...
</description>
```

この例で、makeBooking 操作の入力アクションは明示的に http://example.ibm.com/namespace/makeBooking に定義され、出力アクションは http://example.ibm.com/namespace/makeBookingResponse に定義されています。

詳しくは、W3C WS-Addressing 1.0 Metadata 仕様を参照してください。

WSDL 1.1 のデフォルト・アクション**:**

明示的に指定された <wsa:Action> プロパティーが WSDL 1.1 文書に含まれていない場合、WSDL が DFHWS2LS によって処理されるときに、CICS はデフォルトの入力アクション、出力アクション、および障害アクションを作成します。

WSDL 1.1 のデフォルト入出力アクション

勧告スキーマまたはサブミッション・スキーマに従う WSDL 1.1 文書では、デフォルトの入力または出力アクションを構成するために以下のようなパターンが CICS によって使われます。

```
[target namespace]/[port type name]/[input|output name]
```

WSDL 1.1 のデフォルト障害アクション

勧告スキーマに従う場合、CICS がデフォルトの障害アクションを構築する方法は、スキーマに記述された動作とは異なります。 勧告スキーマに従う WSDL 1.1 文書では、デフォルトの障害メッセージを構成するために以下のようなパターンが CICS によって使われます。 障害の名前が省略されていることに注意してください。

```
[target namespace]/[port type name]/[operation name]/Fault/
```

サブミッション・スキーマに従う場合、 CICS がデフォルトの障害アクションを構築する方法は、スキーマに記述された動作に準拠します。 サブミッション・スキーマに従う WSDL 1.1 文書では、デフォルトの障害メッセージを構成するために以下のようなパターンが CICS によって使われます。

```
[target namespace]/[port type name]/[operation name]/Fault/[fault name]
```

WSDL 1.1 文書用に **CICS** によって生成されたデフォルト・アクションの例

この予約システムの例は、CICS が WSDL 1.1 文書からデフォルト・アクションを構成する方法を示しています。

```
<description targetNamespace="http://example.ibm.com/namespace" ...>
  ...
  <portType name="bookingInterface">
    <operation name="makeBooking">
      <input element="tns:makeBooking" name="MakeBooking"/>
      <output element="tns:bookingResponse" name="BookingResponse"/>
         <fault message="tns:InvalidBooking" name="InvalidBooking"/>
    </operation>
  </interface>
  ...
</definitions>
```

WSDL フラグメントには、以下のアドレス指定プロパティーが含まれています。

プロパティー名	値
[targetNamespace]	`http://example.ibm.com/namespace`
[portType name]	`bookingInterface`
[operation name]	`makeBooking`
[input name]	`MakeBooking`
[output name]	`BookingResponse`
[fault name]	`InvalidBooking`

以下のアクションは、これらの値から作成されます。

アクション	値
入力アクション	`http://example.ibm.com/namespace/bookingInterface/MakeBooking` [input name] が指定されない場合、代わりに "Request" が付加された [operation name] の値が使用されます。 例えば、この場合の入力アクションは `http://example.ibm.com/namespace/bookingInterface/makeBookingRequest` です。

アクション	値
出力アクション	http://example.ibm.com/namespace/bookingInterface/BookingResponse [output name] が指定されない場合、代わりに "Response" が付加された [operation name] の値が使用されます。例えば、この場合の出力アクションは http://example.ibm.com/namespace/bookingInterface/makeBookingResponse です。
障害アクション (勧告スキーマ)	http://example.ibm.com/namespace/bookingInterface/MakeBooking/Fault/ [fault name] が省略されていることに注意してください。
障害アクション (サブミッション・スキーマ)	http://example.ibm.com/namespace/bookingInterface/MakeBooking/Fault/InvalidBooking

詳しくは、W3C WS-Addressing 1.0 Metadata 仕様を参照してください。

WSDL 2.0 のデフォルト・アクション**:**

明示的に指定された <wsa:Action> プロパティーが WSDL 2.0 文書に含まれていない場合、WSDL が DFHWS2LS によって処理されるときに、CICS はデフォルトの入力アクション、出力アクション、および障害アクションを作成します。

WSDL 2.0 のデフォルト入出力アクション

勧告スキーマに従う WSDL 2.0 文書では、入出力のデフォルト・アクションを構成するために以下のようなパターンが CICS によって使われます。

```
[target namespace]/[interface name]/[operation name][direction token]
```

WSDL 2.0 のデフォルト障害アクション

勧告スキーマに従う場合、CICS が WS-Addressing 障害のデフォルト・アクションを構築する方法は、スキーマに記述された動作とは異なります。 サブミッション・スキーマに従う場合、CICS が WS-Addressing 障害のデフォルト・アクションを構築する方法は、スキーマに記述された動作に準拠します。

勧告スキーマに従う WSDL 2.0 文書では、障害のデフォルト・アクションを構成するために以下のようなパターンが CICS によって使われます。 障害の名前が省略されていることに注意してください。

```
[target namespace]/[interface name]/
```

サブミッション・スキーマに従う WSDL 2.0 文書では、障害のデフォルト・アクションを構成するために以下のようなパターンが CICS によって使われます。

```
[target namespace]/[interface name]/[fault name]
```

WSDL 2.0 文書用に **CICS** によって生成されたデフォルト・アクションの例

この例は、CICS が勧告スキーマに従って WSDL 2.0 文書用のデフォルト・アクションを構成する方法を示しています。

```
<description targetNamespace="http://example.ibm.com/namespace" ...>
  ...
  <interface name="bookingInterface">
    <operation name="makeBooking" pattern="http://www.w3.org/ns/wsdl/in-out">
      <input element="tns:makeBooking" messageLabel="In"/>
```

```
      <output element="tns:bookingResponse" messageLabel="Out"/>
    </operation>
  </interface>
  ...
</definitions>
```

WSDL フラグメントには、以下のアドレス指定プロパティーが含まれています。

プロパティー名	値
[targetNamespace]	`http://example.ibm.com/namespace`
[interface name]	`bookingInterface`
[operation name]	`makeBooking`
[direction token]	`Request` または `Response` のどちらか。

以下の入出力アクションは、これらの値から作成されます。

アクション	値
入力アクション	http://example.ibm.com/namespace/bookingInterface/makeBookingRequest
出力アクション	http://example.ibm.com/namespace/bookingInterface/makeBookingResponse

詳しくは、W3C WS-Addressing 1.0 Metadata 仕様を参照してください。

メッセージ交換

Web Services Addressing (WS-Addressing) は、片方向、両方向要求/応答、同期要求/応答、および非同期要求/応答のメッセージ交換をサポートしています。

Web Services Addressing のメッセージ交換には、メッセージ・アドレッシング・プロパティー (MAP) とエンドポイント参照 (EPR) が関係します。

実行時に CICS は、要求メッセージの SOAP ヘッダーに、関連する WS-Addressing メッセージ情報が含まれるようにします。リクエスター・アプリケーションが WS-Addressing ヘッダーを設定する必要はなく、WS-Addressing を使用していることを意識する必要さえないかもしれません。

片方向

この単純な片方向メッセージは、入力のみの操作として定義されます。 この操作を表す Web サービス記述言語 (WSDL) は、次のような形式になります。

```
<operation name="myOperation">
  <input message="tns:myInputMessage"/>
</operation>
```

WS-Addressing を使用している場合、CICS は、実行時に <wsa:Action> MAP および <wsa:MessageID> MAP を WS-Addressing 要求メッセージの SOAP メッセージ・ヘッダーに追加して、確実に WS-Addressing 仕様に準拠するようにします。

<wsa:MessageID> MAP は固有の ID で、指定されない場合は CICS がこの ID を自動的に生成します。

<wsa:Action> MAP は WSDL から派生して WSBind ファイルに保管されます。

これらの MAP の値は、CICS WS-Addressing API コマンドを使って指定変更することができます。

両方向要求/応答

この両方向の交換には、要求メッセージと応答メッセージが関係します。 この操作の応答部分は、出力メッセージ、障害メッセージ、またはその両方として定義可能です。 要求/応答操作を表す WSDL 定義は、次の形式になります。

```
<operation name="myOperation">
  <input message="tns:myInputMessage"/>
  <output message="tns:myOutputMessage"/>
  <fault="tns:myFaultMessage"/>
</operation>
```

エンドポイントに送られる要求に対する応答 (または要求から生成される障害) の宛先は、応答タイプが正常か障害かに応じて、<wsa:ReplyTo> MAP または <wsa:FaultTo> MAP です。

応答の送信先を示すには、<wsa:ReplyTo> または <wsa:FaultTo> MAP を要求メッセージで指定します。

勧告仕様を使用していて、<wsa:ReplyTo> MAP に値を指定しない場合には、<wsa:ReplyTo> MAP はデフォルトの、匿名 URI (http://www.w3.org/2005/08/addressing/anonymous) を含むエンドポイント参照になります。これにより、CICS は応答を要求側に送り戻します。

勧告仕様を使用していて、<wsa:FaultTo> MAP に値を指定しない場合には、<wsa:FaultTo> MAP はデフォルトの、<wsa:ReplyTo> MAP の値になります。

要求側が妥当性検査で不合格となる不正な MAP を作成すると、CICS は不合格のメッセージを <wsa:FaultTo> MAP で指定されたアドレスではなく要求側に送り戻します。

同期要求/応答

デフォルトでは、使用されている基礎プロトコルに従って両方向メッセージの応答部分が戻されます。 HTTP 要求の場合、応答は HTTP 応答で同期的に戻されます。

非同期要求/応答

非同期応答の宛先は別の Web サービスであり、元のリクエスター・アプリケーションに戻されることはありません。 HTTP 要求の場合、要求側のクライアントとの接続は HTTP 202 応答とともに閉じられます。 Web サービス・プロバイダーが CICS システム上で実行されている場合、リクエスター・アプリケーションは空の応答メッセージを受信します。 Web サービス・プロバイダーが WebSphere MQ システム上で実行されている場合、リクエスター・アプリケーションは応答を受信しません。

両方向メッセージの応答部分の宛先を変更するには、適切なアドレスを <wsa:ReplyTo> MAP、または <wsa:ReplyTo> MAP および <wsa:FaultTo> MAP に指定する必要があります。

WSDL 1.1 および WSDL 2.0 で必須の MAP の完全なリストについては、『WS-Addressing の必須のメッセージ・アドレッシング・プロパティー』を参照してください。

WS-Addressing の必須のメッセージ・アドレッシング・プロパティー

WS-Addressing 1.0 メタデータ仕様には、WSDL 1.1 資料および WSDL 2.0 資料に含める必要があるメッセージ・アドレッシング・プロパティー (MAP) が示されています。 WS-Addressing の CICS 実装は、それら必須の MAP に値を自動的に指定することで、WS-Addressing 仕様に準拠する上で役立ちます。

提供する WSDL の MAP に独自の値を指定したり、アドレッシング・コンテキストのそれらの値を CICS WS-Addressing API コマンドを使って更新したりすることができます。 必須の MAP に値を指定しない場合、CICS によって値が自動的に生成されます。

以下の表は、サポートされるさまざまな WSDL 1.1 および WSDL 2.0 メッセージ交換パターン (MEP) の必須の MAP をリストしています。

表 *14. WS-Addressing* の必須のメッセージ・アドレッシング・プロパティー。

WS-Addressing MAP の名前	説明	**WSDL 1.1** で必須	**WSDL 2.0** で必須
<wsa:To>	予期されるメッセージ受信側のアドレス。	No	No
<wsa:Action>	WS-Addressing アクション: 入力、出力、または障害。	以下の MEP で必須: 片方向 両方向 (要求) 両方向 (応答)	以下の MEP で必須: In-only Robust In-only (In) Robust In-only (Fault) In-out (In) In-out (Out) In-optional-out (In) In-optional-out (Out)
<wsa:From>	メッセージの発信元となったエンドポイント。	No	No
この	値	必須では	ありません
<wsa:ReplyTo>	メッセージの応答を受け取ることが予期される受信側のエンドポイント。	No	No
<wsa:FaultTo>	メッセージに関連した障害を受け取ることが予期される受信側のエンドポイント。	No	No
<wsa:MessageID>	固有のメッセージ ID。	以下の MEP で必須: 両方向 (要求)	以下の MEP で必須: Robust In-only (In) In-out (In) In-optional-out (In)

表 14. *WS-Addressing* の必須のメッセージ・アドレッシング・プロパティー。 (続き)

WS-Addressing MAP の名前	説明	**WSDL 1.1** で必須	**WSDL 2.0** で必須
<wsa:RelatesTo>	このメッセージと別のメッセージの関係を示す値のペア。 このエレメントには関連するメッセージの <wsa:MessageID> を組み込み、オプションの属性は関係のタイプを伝送します。	以下の MEP で必須: 両方向 (応答)	以下の MEP で必須: Robust In-only (Fault) In-out (Out) In-optional-out (Out)

詳しくは、「*W3C WS-Addressing 1.0 Metadata*」仕様: http://www.w3.org/TR/ws-addr-metadata/ を参照してください。

注:

- 値が <wsa:ReplyTo> MAP のアドレス・エレメントに設定されない場合、アドレスは匿名 URI `http://www.w3.org/2005/08/addressing/anonymous` に設定されます。 匿名 URI は、応答が要求側に送り戻されることを示します。
- 値が <wsa:FaultTo> MAP のアドレス・エレメントに指定されない場合、 CICS はこのアドレスを <wsa:ReplyTo> MAP のアドレス・エレメントと同じ値に設定します。

  要求側が妥当性検査で不合格となる不正な MAP を作成すると、CICS は不合格のメッセージを <wsa:FaultTo> MAP で指定されたアドレスではなく要求側に送り戻します。
- 値 <wsa:To> MAP が指定されない場合、CICS はアドレスを匿名 URI `http://www.w3.org/2005/08/addressing/anonymous` に設定します。匿名 URI は、DFHWS-URI コンテナーで指定されるアドレスに要求が送信されることを示します。詳しくは、 193 ページの『DFHWS-URI コンテナー』を参照してください。
- WSDL 文書に <wsa:Action> MAP を明示的に定義するか、CICS が自動的に生成するように設定することができます。
- 応答を予期する要求メッセージ、および応答メッセージに関して、 CICS は実行時に、<wsa:MessageID> MAP に対して自動的に固有値を設定します。
- <wsa:RelatesTo> MAP は応答メッセージで必須です。メッセージの関係のタイプはオプションであり、デフォルトは http://www.w3.org/2005/08/addressing/reply になります。

Web Services Addressing のセキュリティー

Web Services Addressing (WS-Addressing) を使って公衆網で伝送される通信を適切に保護し、通信のパーティー間で十分なレベルの信用を確立する必要があります。 通信を保護するために、トランスポート・レベルのセキュリティー (例えば SSL や HTTPS) を使用することをお勧めします。

(SSL や HTTPS などの) トランスポート・レベルのセキュリティーは、WS-Addressing 通信を確実に保護するための最も単純な方法です。 トランスポー

ト・レベルのセキュリティーを使用できない場合、WS-Addressing メッセージ・アドレッシング・プロパティーに署名し、エンドポイント参照を暗号化することによって、メッセージを保護できます。

CICS では WS-Addressing メッセージ・アドレッシング・プロパティーを含むヘッダーを署名することはできず、エンドポイント参照の暗号化も不可能です。 ただし、CICS では着信メッセージの署名を検証したり、暗号化されたヘッダーを暗号化解除することは可能です。 通信を保護するために署名と暗号化を使用したい場合には、外部のセキュリティー・ゲートウェイ (例えば IBM WebSphere DataPower® XML Security Gateway) を使用する必要があります。 詳細については、IBM WebSphere DataPower XML Security Gatewayを参照してください。

Web Services Addressing の例

この例は、メッセージの送信に Web Services Addressing を使用する企業に対して顧客がオーダーを出すときに生じるプロセスに関する概要を示します。

電子部品を販売するある国際企業は、業務に Web Services Addressing を使用しています。 この企業のインフラストラクチャーは、Ordering Client、Distribution Service のグループ、Fulfilment Service、および Configuration Service から構成されます。

WS-Addressing を使用することは、企業にとって以下の利点があります。

- WS-Addressing には、メッセージ転送のためのトランスポートに依存しないメカニズムが備えられています。これにより、異なるプラットフォーム上で実行される Web サービス間の相互運用性が実現されます。 この例では、企業が所有する配布サービスがさまざまなプラットフォームで実行されます。 WS-Addressing を使用することで、異なるプラットフォーム間の相互運用が簡単になります。Web サービスのリクエスターとプロバイダーは、メッセージを交換するサービスが実行されるプラットフォームを意識する必要がないからです。
- WS-Addressing を使用して、<wsa:ReplyTo> MAP 内の EPR を更新することにより、応答メッセージの宛先を変更できます。 この例では、Fulfilment Service がメッセージの転送先の Distribution Service を選択したときに、応答メッセージの宛先を変更します。

企業にはいくつかの異なる国々に複数の配布センターがあります。この例では、各配布センターは Distribution Service によって表されており、 Configuration Service に登録されています。

Fulfilment Service は、例えば要求された品目の在庫や顧客から Distribution Center の距離などのさまざまな要因に基づいて、オーダーを処理するために最適な Distribution Service を選択します。

アドレッシング情報は Configuration Service との間で受け渡されます。 Configuration Service は、使用可能なサービスのアドレスをエンドポイント参照の形式で保管します。 新規のサービスは、**WSAEPR CREATE** コマンドを使用して EPR を作成し、その EPR を Configuration Service に送信することにより、Configuration Service に登録します。 Configuration Service は EPR を XML のブロックとして必要とするため、 DFHWS2LS 上の **WSADDR-EPR-ANY** パラメーター

を TRUE に設定する必要があります。 **WSADDR-EPR-ANY=TRUE** オプションを使用して、 CICS が EPR を <xsd:any> エレメントとして扱うように指示します。CICS は、実行時にそれを言語構造に変換するのではなく、コンテナーに入れる必要があります。

これらのサービスが対話する方法は、以下の図に示されています。 図は、このタスクからは除外されているがビジネス・アプリケーションに関係する可能性がある他のサービスを示しています。これには以下のものがあります。

- 他のそれぞれのサービスによってオーダーの状況とともに更新される可能性があるトラッキング・サービス。
- 発生する障害メッセージを処理するための問題解決サービス。
- Ordering Client に送信される応答メッセージを扱うための Ordering Client コールバック・サービス。

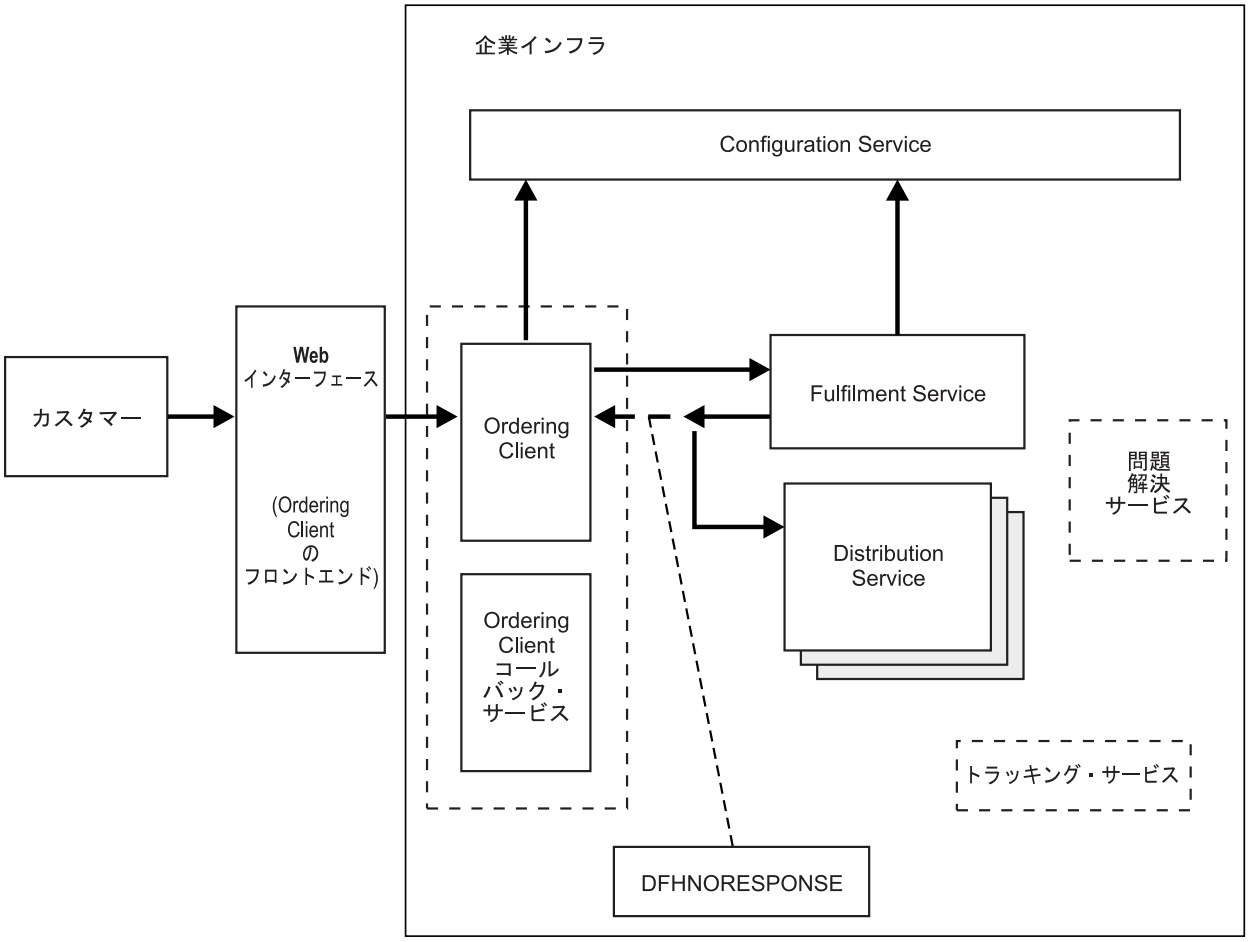


図 *26.* 企業のインフラストラクチャー

以下のステップは、顧客がオーダーを出す時点からそのオーダーが処理される時点までのプロセスを説明しています。

1. 顧客が企業にオーダーを出します。

a. 顧客は Ordering Client のフロントエンドである企業の Web サイトにオーダーを出します。

b. Ordering Client は、顧客の連絡先の詳細をオーダーの一部として取得します。

c. Ordering Client は Web インターフェースを介して確認メッセージおよび固有のオーダー参照を顧客に戻します。

2. Ordering Client はオーダー要求を Fulfilment Service に送信します。

a. Ordering Client が Fulfilment Service の EPR をまだ認識していない場合、 Configuration Service からそれを要求します。 Ordering Client が Configuration Service から Fulfilment Service の EPR を要求する際に関係するプロセスについては、<wsa:To>の例のセクションで詳しく説明されます。

b. Ordering Client は **INVOKE SERVICE** コマンドを Fulfilment Service に対して発行します。 WS-Addressing はメッセージを、要求アドレッシング・コンテキストで EPR によって指定されたアドレスに経路指定します。

3. Fulfilment Service は、オーダーを処理するための Distribution Service を選択し、応答メッセージをそのサービスに転送します。

a. Fulfilment Service は **WSACONTEXT GET** コマンドを使用して、オーダー参照および他のアドレッシング・プロパティーをアドレッシング・コンテキストから抽出します。

b. Fulfilment Service は最適な Distribution Service を Configuration Service から選択します。

c. <wsa:ReplyTo> EPR がアドレッシング・コンテキストに追加されます。

```
<wsa:EndpointReference
     xmlns:wsa="http://www.w3.org/2005/08/addressing">
  <wsa:Address>http://www.example.ibm.com/DistributionService</wsa:Address>
  </wsa:EndpointReference>
```

Fulfilment Service は **WSACONTEXT BUILD** コマンドを使用して、以下のように選択された Distribution Service の ReplyTo EPR を要求アドレッシング・コンテキストに追加します。

d. Fulfilment Service は **WSACONTEXT BUILD** コマンドを繰り返し使用して、オーダー参照や他の情報を要求アドレッシング・コンテキストに追加します。

e. DFHNORESPONSE コンテナーが Ordering Client パイプラインに追加されて、応答を受け取らないことを Ordering Client に知らせます。応答メッセージは、要求メッセージの形式で Distribution Service に転送されます。

4. Distribution Service は、転送された応答メッセージを受け取り、オーダーを処理します。

a. Distribution Service は **WSACONTEXT GET** コマンドを使用して、オーダー参照およびアドレッシングの詳細を要求アドレッシング・コンテキストから抽出します。

b. Distribution Service はオーダーを処理します。

<wsa:To> の例

1. Ordering Client は、メッセージの送信先とするサービスの EPR を Configuration Service から要求します。 この例では、Ordering Client は Fulfilment Service の EPR を要求します。
2. Configuration Service は、次のように応答メッセージの作成および送信を行います。
   a. Configuration Service は、**WSAEPR CREATE** API コマンドを使用して、Fulfilment Service の要求された <wsa:To> EPR を作成します (EXEC CICS WSAEPR CREATE)。
   b. Configuration Service は **WSAEPR CREATE** コマンドの出力をコンテナーに書き込みます (EXEC CICS PUT CONTAINER(work-cont))。
   c. Configuration Service はコンテナー名を myEpr-xml-contエレメントにコピーします (MOVE work-cont TO myEpr-xml-cont)。
   d. Configuration Service は、応答メッセージを Ordering Client に送信します。このメッセージには、myEpr-xml-cont コンテナーによって指定されたコンテナーの内容が含まれています。 この例では、work-cont コンテナーの内容は、<wsa:myEpr> エレメント内の Ordering Client に送信されます。

```
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
   ...
   <env:Body>
      <wsa:myEpr>
         <wsa:EndpointReference>
            <wsa:Address>
          Fulfilment_Service_EPR_XML
            </wsa:Address>
         </wsa:EndpointReference>
      </wsa:myEpr>
   </env:Body>
   ...
</env:Envelope>
```

263 ページの図 27 は、Ordering Client と Configuration Service の間の要求/応答メッセージの交換が示されています。 このメッセージ交換には、2 つの標準的な Web サービス・パイプラインが関係します。

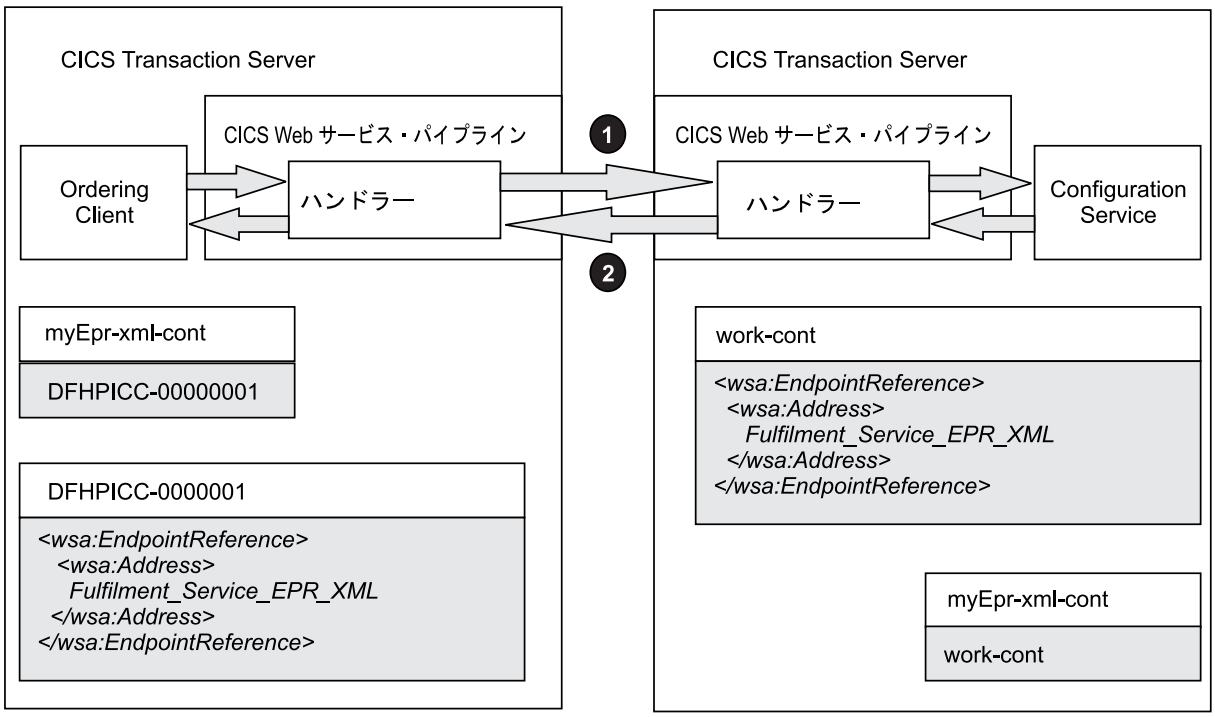


図 27. *Ordering Client* と *Configuration Service* の間の要求/応答メッセージの交換

3. Ordering Client は、応答メッセージを受け取り、<wsa:To> EPR を作成し、要求を Fulfilment Service に送信します。
   a. Ordering Client は <wsa:To> EPR データを応答メッセージから抽出します。
   b. CICS は固有のコンテナー (この例では `DFHPICC-00000001` コンテナー) に <wsa:To> EPR データを取り込みます。
   c. CICS はコンテナーの名前 (この例では `DFHPICC-00000001`) を `myEpr-xml-cont` エレメントにコピーします。
   d. Ordering Client は `myEpr-xml-cont` エレメントによって指定されるコンテナーの内容を読み取り、それを **`WSACONTEXT BUILD`** API コマンドに入力として提供します。 **`WSACONTEXT BUILD`** コマンドはこの入力を使用して、Fulfilment Service の <wsa:To> EPR を作成します。
   e. Ordering Client は、パイプライン処理を開始する **`INVOKE SERVICE`** コマンドを発行します。
   f. アウトバウンド・パイプライン上の CICS Web サービスのアドレッシング・ハンドラー DFHWSADH は <wsa:To> EPR をアドレスおよびオプションの一連の参照パラメーターに変換し、Fulfilment Service に送信される SOAP 要求メッセージのヘッダーにそれを置きます。

```
<env:Header>
  <wsa:To>http://example.ibm.com/Fulfilment_Service</wsa:To>
</env:Header>
```

264 ページの図 28 は、Ordering Client から Fulfilment Service への要求を示します。この要求には、CICS Web サービスのアドレッシング・ハンドラー

DFHWSADH を組み込む Web サービス・パイプラインが関係します。

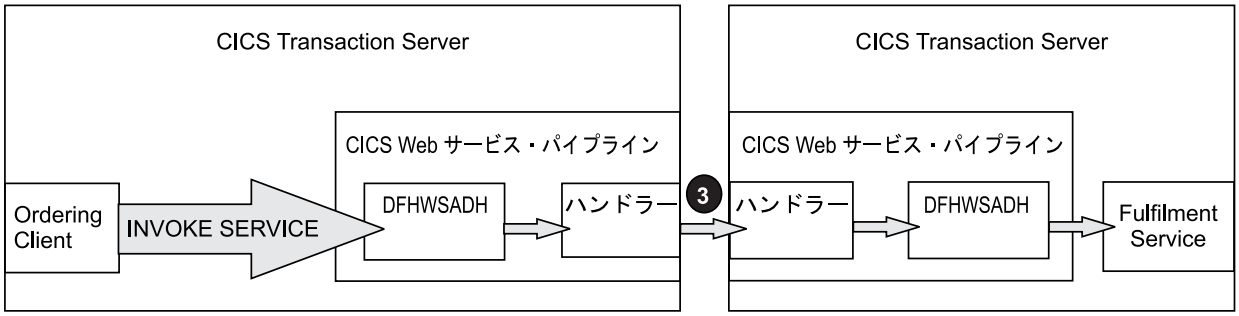


図 *28. Ordering Client* から *Fulfilment Service* への要求

Web Services Addressing の用語

Web Services Addressing (WS-Addressing) のサポートについて説明する際に使用される用語。

アドレッシング・コンテキスト **(addressing context)**
: WS-Addressing メッセージ・アドレッシング・プロパティー (MAP) が SOAP 要求メッセージに送信される前と SOAP 要求メッセージおよび応答メッセージから受信された後にそれを保管する XML 文書。

エンドポイント参照 **(EPR) (endpoint reference (EPR))**
: メッセージを Web サービスに経路指定するのに使用されるアドレッシング情報を含む XML 構造。 このアドレッシング情報には、メッセージの宛先アドレス、アプリケーションが使用するオプションの参照パラメーター、およびオプションのメタデータが含まれます。

メッセージ・アドレッシング・プロパティー **(MAP) (message addressing property (MAP))**
: 固有のメッセージ ID、メッセージの宛先、およびメッセージのエンドポイント参照など、特定の Web サービス・メッセージに関するアドレッシング情報を伝達する XML エレメント。

SAML のサポート

CICS は、オンライン・ビジネス・パートナー間でセキュリティー情報を記述し交換するための Security Assertion Markup Language (SAML) の使用をサポートします。

CICS は SAMLCore1.1 および SAML Core2.0 標準をサポートします。これらの標準に記述されているプロトコルはサポートしません。

SAML トークンを使用するようにプロバイダーおよびリクエスター・パイプラインを構成することができますが、最初に CICS セキュリティー・トークン・サービス

(STS) を配置する必要があります。SAML をサポートするように CICS インストールを構成する方法に関する詳細については、SAML 用の CICS の構成を参照してください。

第 3 章 Web サービスの開発

JSON Web サービスの作成

既存の CICS アプリケーションを JSON Web サービスとして公開し、新規の CICS アプリケーションを作成して JSON Web サービス・プロバイダーとして機能させることができます。

始める前に

JSON Web サービスの作成を始める前に、JSON Web サービスをサポートするよう CICS システムを構成する必要があります。詳しくは、 JSON 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を参照してください。

このタスクについて

付属のユーティリティーである CICS JSON アシスタントは、新しい JSON Web サービス・プロバイダー・アプリケーション用の必要な成果物を作成したり、既存のアプリケーションを JSON Web サービス・プロバイダーとして使用可能にしたりする作業を支援します。

CICS JSON アシスタントは高水準言語構造からの JSON スキーマの作成、および既存の JSON スキーマからの高水準言語構造の作成を行うことができ、COBOL、C/C++、PL/I をサポートします。また、JSON メッセージからコンテナーおよび COMMAREA への (さらにこの逆方向への) 自動ランタイム変換を可能にするための情報も生成します。この情報は、パイプラインの処理中に、CICS JSON Web サービス・サポートにより使用されます。

以下の手順の記述に従って JSON Web サービスを作成した後、正しく機能することを検証します。

手順

1. JSON Web サービスを作成します。 JSON アシスタントを使用して JSON スキーマまたは言語構造を作成し、それらを CICS に配置します。 **PIPELINE SCAN** コマンドを使用して、必要な CICS リソースを自動的に作成します。
2. JSON Web サービスを開始して、想定どおりに機能することをテストします。

次のタスク

これらのステップの詳細については、次のトピックで説明します。

CICS JSON アシスタント

CICS JSON アシスタントは、JSON スキーマと言語構造の間のマッピングを作成するバッチ・ユーティリティー・セットです。このマッピングは、JSON とアプリケーション・データの間の変換を行うために CICS によって実行時に使用されます。

© Copyright IBM Corp. 2005, 2018

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 JSON 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、JSON 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でデータをマップするからです。

このアシスタントは高水準言語構造からの JSON スキーマの作成、および既存の JSON スキーマからの高水準言語構造の作成を行うことができ、
COBOL、C/C++、PL/I をサポートします。また、JSON メッセージからコンテナーおよび COMMAREA への (さらにこの逆方向への) 自動ランタイム変換を可能にするための情報も生成します。

CICS JSON 支援機能は以下の 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。

DFHLS2JS
: 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、JSON スキーマも生成します。

DFHJS2LS
: JSON スキーマを基にして 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。

DFHLS2JS または DFHJS2LS プロシージャーに関連する使用モードは、要件に応じて次のように異なります。

- DFHLS2JS: リンク可能インターフェースに関する高水準言語から JSON スキーマへの変換
- DFHJS2LS: リンク可能インターフェースに関する JSON スキーマから高水準言語への変換
- DFHLS2JS: 要求/応答サービスに関する高水準言語から JSON スキーマへの変換
- DFHJS2LS: 要求/応答サービスに関する JSON スキーマから高水準言語への変換
- DFHJS2LS: RESTful サービスにおける JSON スキーマから高水準言語への変換

JSON アシスタント・ユーティリティー・プログラムとデータ・マッピングの詳細については、以下のトピックを参照してください。

DFHLS2JS: 要求/応答サービスに関する高水準言語から JSON スキーマへの変換

DFHLS2JS プロシージャーは、高水準言語データ構造から JSON スキーマ・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、DFHLS2JS を使用することができます。

DFHLS2JS JCL プロシージャーは、データ・セット *HLQ*.XDFHINST にインストールされます。ここで、*HLQ* は、CICS がインストールされている高位修飾子です。

DFHLS2JS 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 (DFHLS2JS) を指定します。

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

DFHLS2JS では以下のシンボリック・パラメーターが定義されています。

JAVADIR = *path*
DFHLS2JS 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/ に付加されて、/usr/lpp/*path* という完全なパス名が生成されます。

通常、このパラメーターは指定しません。 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。

PATHPREF = *prefix*
他のパラメーターで使用される z/OS UNIX ディレクトリー・パスを拡張する接頭部を指定します。接頭部を使用しない場合は、'' (空ストリング) を指定します。

通常、このパラメーターは指定しません。 デフォルト値は、**PATHPREF** 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。

SERVICE = *value*
このパラメーターは IBM サポートに指示された場合にのみ使用します。

TMPDIR = *tmpdir*
DFHLS2JS が一時ワークスペースとして使用する z/OS UNIX のディレクトリーの場所を指定します。 ジョブの実行に使用されるユーザー ID には、このディレクトリーへの読み取り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*
DFHLS2JS が一時ワークスペース・ファイルの名前を構成するために使用する接頭部を指定します。

デフォルト値は LS2JS です。

USSDIR = *path*

UNIX システム・サービスのファイル・システム内の 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/ に付加されて、/usr/lpp/cicsts/*path* という完全なパス名が生成されます。デフォルトを使用する場合は、これを '.' (ピリオド) として指定する必要があります。

通常、このパラメーターは指定しません。 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。

一時ワークスペース

DFHLS2JS により、実行時に以下の 3 つの一時ファイルが作成されます。

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。

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

/tmp/LS2JS.in

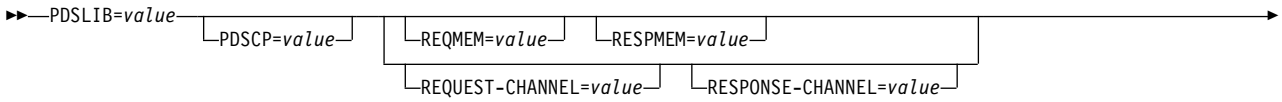
/tmp/LS2JS.out

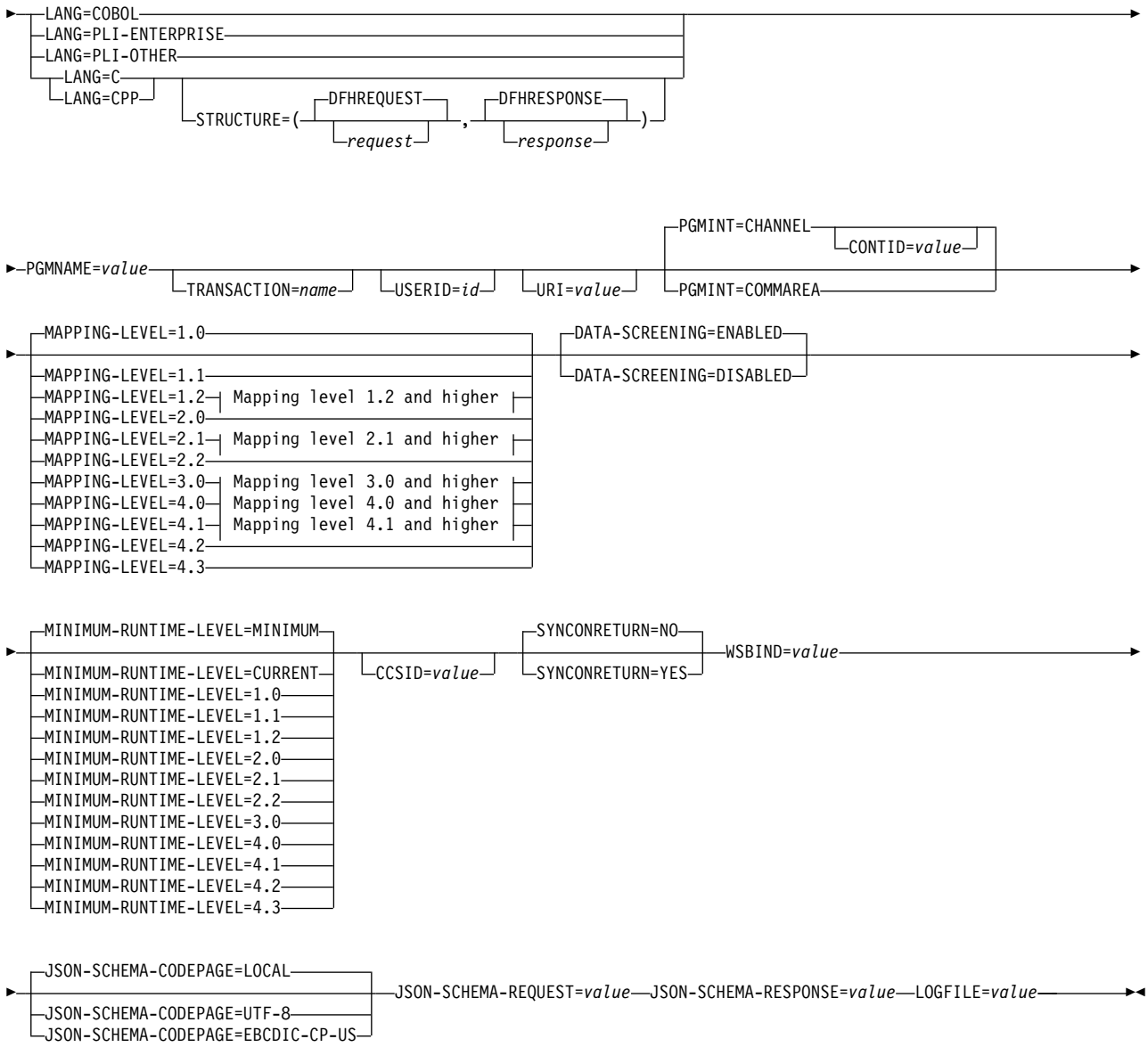
/tmp/LS2JS.err

重要: DFHLS2JS は、z/OS UNIX ファイルまたはデータ・セット・メンバーへのアクセスをロックしません。DFHLS2JS の複数のインスタンスが同時に実行され、同じ一時ワークスペース・ファイルを使用する場合、あるジョブがワークスペース・ファイルを使用している間に別のジョブがそれらのファイルを上書きするのを防ぐことはできません。そのため、予測不能な障害が発生する可能性があります。

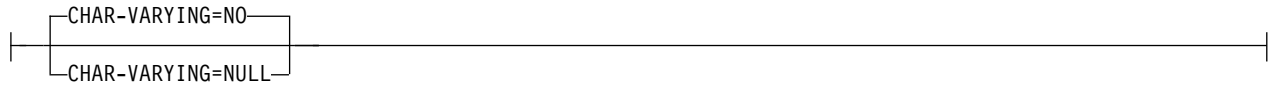
したがって、この状況を回避する命名規則および操作手順を考案することをお勧めします。例えば、システム・シンボリック・パラメーター **SYSUID** を使用して、個々のユーザー固有のワークスペース・ファイル名を生成できます。これらの一時ファイルはジョブの終わりの前に削除されます。

DFHLS2JS の入力パラメーター

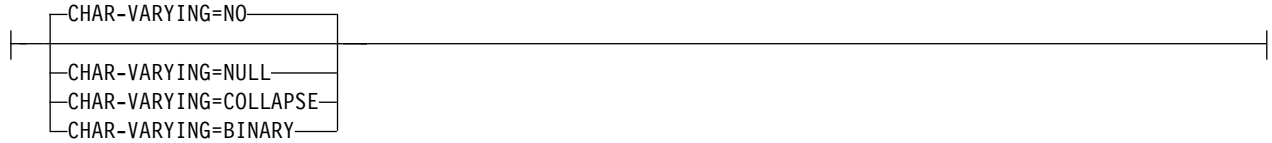




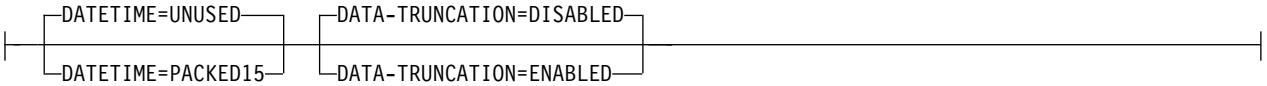
マッピング・レベル **1.2** 以上の場合:



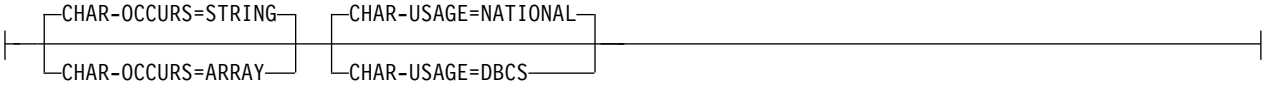
マッピング・レベル **2.1** 以上の場合:



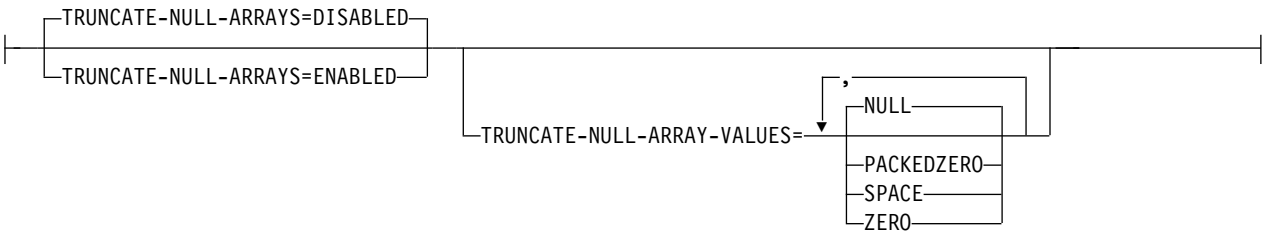
マッピング・レベル **3.0** 以上**:**



マッピング・レベル **4.0** 以上の場合**:**



マッピング・レベル **4.1** 以上の場合**:**



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。
- パラメーターおよび使用する場合は継続文字は、72 列を超えてはなりません。 列 73 から 80 はブランクにする必要があります。
- 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。 スペースを含むアスタリスクより前の文字はすべてパラメーターの一部とみなされます。 以下に例を示します。

  ```
  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。
- 行の末尾の文字の位置にコンマがある場合、これはオプションの行分離文字であり、無視されます。

パラメーターの記述

CCSID = *value*

アプリケーション・データ構造に文字データをエンコードするために、実行時に使用される CCSID を指定します。このパラメーターの値は、**LOCALCCSID** システム初期設定パラメーターの値を指定変更します。 *value* は、Java および z/OS Unicode Services ユーザーズ・ガイドおよび解説書でサポートされる

EBCDIC CCSID である必要があります。このパラメーターを指定しない場合、アプリケーション・データ構造は、システム初期設定パラメーターで指定された CCSID を使用してエンコードされます。

CHAR-VARYING = { **NO** | **NULL** | **COLLAPSE** | **BINARY** }

マッピング・レベルが 1.2 以上のとき、言語構造内の文字フィールドがマップされる方法を指定します。 COBOL の文字フィールドは、タイプ X のピクチャー節 (例えば PIC(X) 10) です。C/C++ の文字フィールドは文字配列です。以下のオプションを選択できます。

NO 文字フィールドは JSON ストリングにマップされ、固定長のフィールドとして処理されます。 データの最大長はフィールドの長さと同じです。NO は、マッピング・レベル 2.0 以前での、COBOL および PL/I の **CHAR-VARYING** パラメーターのデフォルト値です。

この値は、Enterprise およびその他の PL/I 言語構造に適用されません。

NULL

文字フィールドは JSON ストリングにマップされ、ヌル終了ストリングとして処理されます。CICS は、JSON メッセージから変換する際に、終了ヌル文字を追加します。 文字ストリングの最大長は、言語構造に示される長さより 1 文字少ない長さとして計算されます。 NULL は、C/C++ での **CHAR-VARYING** パラメーターのデフォルト値です。

この値は、Enterprise およびその他の PL/I 言語構造に適用されません。

COLLAPSE

文字フィールドは、JSON ストリングにマップされます。フィールド内の後続空白と埋め込み空白は JSON メッセージに含まれません。例えば、<space>AB<space><space><space>C<space> は AB<space>C になります。インバウンド JSON メッセージは解析され、先行空白、後続空白、埋め込み空白はすべて削除されます。マッピング・レベル 2.1 以降の COBOL および PL/I では、**CHAR-VARYING** パラメーターのデフォルト値は COLLAPSE です。

BINARY

文字フィールドは、base64 エンコード・データを含む JSON ストリングにマップされ、固定長のフィールドとして処理されます。
CHAR-VARYING パラメーターで BINARY 値が使用できるのは、マッピング・レベル 2.1 以降のみです。

可変長の値と空白の処理についての詳細は、 411 ページの『可変長の値と空白のサポート』を参照してください。

CHAR-OCCURS = { **STRING** | **ARRAY** }

マッピング・レベル 4.0 以上の場合に、言語構造内の文字配列をマップする方法を指定します。例えば PIC X OCCURS 20 となります。このパラメーターは COBOL 言語でのみ使用されます。

ARRAY

文字配列は JSON 配列にマップされます。つまり、それぞれの文字が

個別の JSON エレメントとしてマップされます。マッピング・レベル 3.0 以前でも、このように動作します。

STRING
文字配列は JSON ストリングにマップされます。つまり、COBOL 配列全体が 1 つの JSON エレメントとしてマップされます。

CHAR-USAGE = { **NATIONAL** | **DBCS** }

COBOL では、各国語データ型 PIC N を UTF-16 または DBCS データ用に使用できます。この設定は NSYMBOL コンパイラー・オプションによって制御されます。データが適切に扱われるようにするには、アシスタントの **CHAR-USAGE** パラメーターを NSYMBOL コンパイラー・オプションと同じ値に設定する必要があります。 UTF-16 を使用する場合には、通常、これは CHAR-USAGE=NATIONAL に設定されます。

DBCS
PIC (*n*) フィールドからのデータは、DBCS でエンコードされたデータとして扱われます。

NATIONAL
PIC (*n*) フィールドからのデータは、UTF-16 でエンコードされたデータとして扱われます。

CONTID = *value*
サービス・プロバイダーで、JSON メッセージを表すために使用される最上位のデータ構造を保持するコンテナーの名前を指定します。

CICS がターゲット・アプリケーション・プログラムに渡すコンテナーの長さは、要求コンテナーの長さおよび応答コンテナーの長さより大きくなります。

DATA-SCREENING = { **ENABLED** | **DISABLED** }
アプリケーション提供のデータのエラーを検査するかどうかを指定します。

ENABLED
言語構造と矛盾するアプリケーション提供のランタイム・データは、エラーとして扱われ、メッセージ DFHPI1010 が発行されます。エラー応答がアプリケーションに 返されます。

DISABLED
言語構造と矛盾するアプリケーション提供のランタイム・データの値は、 デフォルト値で置き換えられます。例えば、数値フィールド内のゼロは誤った値を置き換えます。メッセージ DFHPI1010 は発行されず、通常の応答がアプリケーションに返されます。この機能を使用して、 初期設定されていない出力フィールドから生成される INVALID_PACKED_DEC および INVALID_ZONED_DEC エラー応答を回避できます。

DATA-TRUNCATION = { **DISABLED** | **ENABLED** }
固定長フィールド構造で可変長データを許容するかどうかを指定します。

DISABLED
データが CICS が予期する固定長より短い場合に、CICS は切り捨てられたデータを拒否してエラー・メッセージを発行します。

ENABLED
データが CICS が予期する固定長より短い場合に、CICS は切り捨てられたデータを許容して、欠落データをヌル値として処理します。

DATETIME = { **UNUSED** | **PACKED15** }
高水準言語構造内で ABSTIME になる可能性のあるフィールドをタイム・スタンプとしてマップするかどうかを指定します。

PACKED15
長さ 15 (8 バイト) のパック 10 進数フィールドは CICS の ABSTIME フィールドとして扱われ、タイム・スタンプとしてマップされます。

UNUSED
長さ 15 (8 バイト) のパック 10 進数フィールドは、タイム・スタンプとしては扱われません。

このパラメーターは、マッピング・レベル 3.0 で設定できます。

JSON-SCHEMA-CODEPAGE = { **LOCAL** | **UTF-8** | **EBCDIC-CP-US** }
JSON スキーマ文書を生成するために使用されるコード・ページを指定します。

LOCAL
ファイル・システムのデフォルトのコード・ページを使用して生成された JSON スキーマを指定します。

UTF-8
UTF-8 コード・ページを使用して生成された JSON スキーマを指定します。

EBCDIC-CP-US
US EBCDIC コード・ページを使用して生成された JSON スキーマを指定します。

JSON-SCHEMA-REQUEST = *value*
これは必須パラメーターです。

この値は、要求 JSON スキーマが格納される UNIX システム・サービスの場所を示します。

JSON-SCHEMA-RESPONSE = *value*
これは必須パラメーターです。

この値は、応答 JSON スキーマが格納される UNIX システム・サービスの場所を示します。

LANG = **COBOL**|**PLI-ENTERPRISE**|**PLI-OTHER**|**C**|**CPP**
高水準言語構造のプログラミング言語を指定します。

COBOL
COBOL

PLI-ENTERPRISE
Enterprise PL/I

PLI-OTHER
Enterprise PL/I 以外の PL/I のレベル

C C

CPP C++

LOGFILE = *value*

DFHLS2JS 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 z/OS UNIX 名です。 DFHLS2JS はこのファイルを作成しますが、存在していないディレクトリー構造を作成することはありません。

通常はこのファイルを使用しませんが、DFHLS2JS 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。

MAPPING-LEVEL = **{ 1.0 | 1.1 | 1.2 | 2.0 | 2.1 | 2.2 | 3.0 | 4.0 | 4.1 | 4.2 | 4.3 }**

Web サービス・バインディング・ファイルおよび JSON スキーマを生成する際に、DFHLS2JS が使用するマッピングのレベルを指定します。以下のオプションを選択できます。

1.0 このオプションは、SOAP Web サービスとの互換性のために保持されています。 JSON で使用することは推奨されていません。

1.1 このオプションは、SOAP Web サービスとの互換性のために保持されています。 JSON で使用することは推奨されていません。

1.2 このオプションは、SOAP Web サービスとの互換性のために保持されています。 JSON で使用することは推奨されていません。

2.0 このオプションは、SOAP Web サービスとの互換性のために保持されています。 JSON で使用することは推奨されていません。

2.1 このオプションは、SOAP Web サービスとの互換性のために保持されています。 JSON で使用することは推奨されていません。

2.2 このオプションは、SOAP Web サービスとの互換性のために保持されています。 JSON で使用することは推奨されていません。

3.0 このマッピング・レベルは、使用可能なすべてのオプション・セットを使用して JSON スキーマを生成する場合に使用します。

4.0 このマッピング・レベルは、CICS TS 5.2 以降の領域で使用します。このマッピング・レベルでは、COBOL の OCCURS DEPENDING ON フィールドおよび **CHAR-OCCURS** パラメーターを使用できます。

4.1 切り捨て可能な配列をサポートするには、CICS TS 5.2 以降の領域でこのマッピング・レベルを使用します。

4.2 大きな変更はありません。このマッピング・レベルは、CICS TS 5.4 以降の領域で使用します。

4.3 大きな変更はありません。このマッピング・レベルは、CICS TS 5.4 以降の領域で使用します。

マッピング・レベルの詳細については、『CICS JSON 支援機能用のマッピング・レベル』を参照してください。

MINIMUM-RUNTIME-LEVEL = **{ MINIMUM | 1.0 | 1.1 | 1.2 | 2.0 | 2.1 | 2.2 | 3.0 | 4.0 | 4.1 | 4.2 | 4.3 | CURRENT }**

Web サービス・バインディング・ファイルを配置できる最小の CICS 実行時環境を指定します。指定した他のパラメーターと一致しないレベルを選択すると、エラー・メッセージが送信されます。 以下のオプションを選択できます。

MINIMUM
選択したパラメーターを前提として、最低限可能な CICS 実行時レベルが自動的に割り振られます。

1.0 このオプションは、SOAP Web サービスとの互換性のために保持されています。 JSON で使用することは推奨されていません。

1.1 このオプションは、SOAP Web サービスとの互換性のために保持されています。 JSON で使用することは推奨されていません。

1.2 このオプションは、SOAP Web サービスとの互換性のために保持されています。 JSON で使用することは推奨されていません。

2.0 このオプションは、SOAP Web サービスとの互換性のために保持されています。 JSON で使用することは推奨されていません。

2.1 このオプションは、SOAP Web サービスとの互換性のために保持されています。 JSON で使用することは推奨されていません。

2.2 このオプションは、SOAP Web サービスとの互換性のために保持されています。 JSON で使用することは推奨されていません。

3.0 生成された Web サービス・バインディング・ファイルは、CICS TS 4.1 以降の領域にデプロイされます。

注: JSON サポートが使用可能なのは、CICS TS 4.2 以降に限られます。

4.0 生成された Web サービス・バインディング・ファイルは、CICS TS 5.2 以降の領域に正常にデプロイされます。この実行時レベルでは、**MAPPING-LEVEL** パラメーターにマッピング・レベル 4.0 以前を使用できます。このレベルでは任意のオプション・パラメーターを使用できます。

4.1 生成された Web サービス・バインディング・ファイルは、CICS TS 5.2 以降の領域に正常にデプロイされます。この実行時レベルでは、**MAPPING-LEVEL** パラメーターには、マッピング・レベル 4.1 以前を使用することができます。

4.2 生成された Web サービス・バインディング・ファイルは、CICS TS 5.4 以降の領域に正常にデプロイされます。この実行時レベルでは、**MAPPING-LEVEL** パラメーターには、マッピング・レベル 4.2 以前を使用することができます。

4.3 生成された Web サービス・バインディング・ファイルは、CICS TS 5.4 以降の領域に正常にデプロイされます。この実行時レベルでは、**MAPPING-LEVEL** パラメーターには、マッピング・レベル 4.3 以前を使用することができます。

CURRENT
生成された Web サービス・バインディング・ファイルは、Web サービス・バインディング・ファイルの生成に使用するものと同じ実行時レベルで、CICS 領域に正常に配置されます。

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、**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。

PDSCP = *value*

REQMEM および **RESPMEM** パラメーターに指定される区分データ・セット・メンバーで使用されるコード・ページを指定します。ここで、*value* は CCSID 番号または Java コード・ページ番号です。このパラメーターを指定しない場合は、z/OS UNIX システム・サービスのコード・ページが使用されます。 例えば、**PDSCP**=037 と指定できます。

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ターゲット・アプリケーション・プログラムにデータを渡します。

- マッピング・レベル 3.0 より前では、チャネルに含められるのは 1 つのコンテナーのみで、このコンテナーは入出力の両方に使用されます。**CONTID** パラメーターを使用してコンテナーの名前を指定します。 デフォルト名は DFHWS-DATA です。
- マッピング・レベル 3.0 では、チャネルに複数のコンテナーを含めることができます。 **REQUEST-CHANNEL** および **RESPONSE-CHANNEL** パラメーターを使用します。 **PDSLIB**、**REQMEM**、および **RESPMEM** は指定しないでください。

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。

ターゲット・アプリケーション・プログラムが要求を処理するとき、同じメカニズムを使用して応答を戻す必要があります。要求が通信域で受信されている場合は、応答を通信域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、応答をコンテナーに戻す必要があります。CICS がターゲット・アプリケーション・プログラムに渡す通信域またはコンテナーの長さは、要求の通信域またはコンテナーの長さ、および応答の通信域またはコンテナーの長さより大きくなります。

PGMNAME = *value*

Web サービスとして公開されるターゲット・アプリケーション・プログラムの、CICS PROGRAM リソースの名前を指定します。 CICS Web サービス・サポートは、このプログラムにリンクします。

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。

REQUEST-CHANNEL = *value*

チャネル記述文書の名前と場所を指定します。チャネル記述は、Web サービ

ス・リクエスターから JSON メッセージを受信する際に Web サービス・プロバイダー・アプリケーションがそのインターフェースで使用できるコンテナーについて記述します。チャネル記述は、CICS 提供のチャネル・スキーマに準拠する必要がある XML 文書です。詳しくは、 317 ページの『チャネル記述文書の作成』を参照してください。

このパラメーターはマッピング・レベル 3.0 のみで使用できます。

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。

RESPONSE-CHANNEL = *value*
チャネル記述文書の名前と場所を指定します。チャネル記述は、Web サービス・リクエスターに JSON 応答メッセージを送信する際に Web サービス・プロバイダー・アプリケーションがそのインターフェースで使用できるコンテナーについて記述します。チャネル記述は、CICS 提供のチャネル・スキーマに準拠する必要がある XML 文書です。詳しくは、 317 ページの『チャネル記述文書の作成』を参照してください。

このパラメーターはマッピング・レベル 3.0 のみで使用できます。

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や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。

応答
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や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。

SYNCONRETURN = { **NO** | **YES** }
リモート Web サービスが同期点を発行できるかどうかを指定します。

NO リモート Web サービスは同期点を発行できません。 この値はデフォルトです。リモート Web サービスが同期点を発行すると、ADPL 異常終了になり失敗します。

YES リモート Web サービスは同期点を発行できます。 YES を選択した場合、リモート Web サービスから制御が戻されたときにリモート・タスクが別個の作業単位としてコミットされます。リモート Web サービスがリカバリー可能リソースを更新して戻した後に障害が発生した場合、そのリソースの更新内容をバックアウトすることはできません。

TRANSACTION = *name*

サービス・プロバイダーで、このパラメーターは、応答を組み立てるためにパイプラインを開始できる 1 から 4 文字の別名トランザクションの名前を指定します。 このパラメーターの値は、URIMAP リソースが **PIPELINE** スキャン・コマンドを使用して自動的に作成される際に、URIMAP リソースの TRANSACTION 属性を定義するために使用されます。

許容文字:
A-Z a-z 0-9 $ @ # _ < >

TRUNCATE-NULL-ARRAYS = { **DISABLED** | **ENABLED** }

マッピング・レベル 4.1 以上で構造化配列を処理する方法を指定します。この機能を有効にすると、CICS は配列に含まれる空のレコードを認識しようと試みます (空のレコードの認識について詳しくは、TRUNCATE-NULL-ARRAY-VALUES を参照)。空の配列レコードが 5 個連続して検出された場合、その配列は XML/JSON の生成時に最初の空のレコードの箇所で切り捨てられます。この機能は、内容が構造化されている配列に対してのみ有効にされます。単純なプリミティブ・フィールドからなる配列には、切り捨ては適用されません。配列の切り捨てにより、JSON/XML 内のデータ表現が簡潔になりますが、リスクがないわけではありません。5 個の連続したデータ・レコードが、(おそらく、正当な低値を含んでいるために) 初期化されていないストレージとして誤って認識された場合、データ損失が生じるおそれがあります。TRUNCATE-NULL-ARRAYS が有効にされていて、TRUNCATE-NULL-ARRAY-VALUES が設定されていない場合は、TRUNCATE-NULL-ARRAY-VALUES のデフォルト値が使用されます。

| **TRUNCATE-NULL-ARRAY-VALUES** = { **NULL** | **PACKEDZERO**|**SPACE** | **ZERO** }

マッピング・レベル 4.1 以上で、TRUNCATE-NULL-ARRAYS の処理で空として扱う値を指定します。デフォルトでは、ヌル値 (0x00、または小さい値) が空として扱われます。構造化された配列のレコードに含まれるすべてのストレージ・バイトにヌルが含まれている場合、レコード全体が空であると見なされます。1 つ以上の NULL、PACKEDZERO、SPACE、および ZERO 値をコンマ区切りリストに指定できます。

NULL
ヌル文字 (0x00) を暗黙指定します。

PACKEDZERO
正符号付きパック 10 進数ゼロ (0x0C)、負符号付きパック 10 進数ゼロ (0x0D)、または符号なしパック 10 進数ゼロ (0x0F) を暗黙指定します。

SPACE
SBCS EBCDIC スペース (0x40) を暗黙指定します。

ZERO
符号なしのゾーン 10 進数ゼロ (0xF0) を暗黙指定します。

構造化配列レコードに、選択したバイトの組み合わせとの一致が含まれている場合、レコード全体が空として識別されます。

TRUNCATE-NULL-ARRAY-VALUES に値が定義されている場合、
TRUNCATE-NULL-ARRAYS が有効に設定されている必要があります。

URI = *value*
このパラメーターは、クライアントが Web サービスにアクセスするときに使用する相対または絶対 URI を指定します。 CICS は、DFHLS2JS 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 のパスのコンポーネントを指定します。

USERID = *id*
サービス・プロバイダーでは、このパラメーターは、任意の Web クライアントで使用可能な 1 文字から 8 文字のユーザー ID を指定します。 アプリケーションから生成される応答や Web サービスについては、そのユーザー ID の下で別名トランザクションが追加されます。 このパラメーターの値は、URIMAP リソースが **PIPELINE** スキャン・コマンドを使用して自動的に作成される際に、URIMAP リソースの USERID 属性を定義するために使用されます。

許容文字:
A-Z a-z 0-9 $ @ #

WSBIND = *value*
Web サービス・バインディング・ファイルの完全修飾 z/OS UNIX 名です。 DFHLS2JS はこのファイルを作成しますが、存在していないディレクトリー構造を作成することはありません。 ファイル拡張子は .wsbind です。

その他の情報

- DFHLS2JS の実行に使用するユーザー ID は、UNIX システム・サービスを使用するように構成されていなければなりません。このユーザー ID には、CICS z/OS UNIX ファイル構造および PDS ライブラリーに対する読み取り権限と **LOGFILE**、**WSBIND**、および **JSON Schema** 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。
- JCL の最大パラメーター長は 100 文字です。 このパラメーター長は、**STDPARM** ステートメントを使用して増やすことができます。詳しくは、「z/OS UNIX システム・サービス ユーザーズ・ガイド」を参照してください。

例

```
//LS2JS JOB '
accounting information
',
name,MSGCLASS=A
// SET QT=''''
//JAVAPROG EXEC DFHLS2JS,
// TMPFILE=&amp;QT.&SYSUID.&QT,
//INPUT.SYSUT1 DD *
PDSLIB=CICSHLQ.SDFHSAMP
REQMEM=DFH0XCP4
RESPMEM=DFH0XCP4
JSON-SCHEMA-REQUEST=/u/exampleapp/json/example_request.json
JSON-SCHEMA-RESPONSE=/u/exampleapp/json/example_response.json
```

```
LANG=COBOL
LOGFILE=/u/exampleapp/wsbind/example.log
MAPPING-LEVEL=4.0
CHAR-VARYING=COLLAPSE
PGMNAME=DFH0XCMN
URI=http://myserver.example.org:8080/exampleApp/example
PGMINT=COMMAREA
SYNCONRETURN=YES
WSBIND=/u/exampleapp/wsbind/example.wsbind
/*
```

DFHJS2LS: 要求/応答サービスに関する JSON スキーマから高水準言語への変換

DFHJS2LS プロシージャーは JSON スキーマから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。 DFHJS2LS は、サービス・プロバイダーとして CICS アプリケーション・プログラムを作成するときに準備段階で使用できます。このトピックには、DFHJS2LS のジョブ制御ステートメント、シンボリック・パラメーター、および入力パラメーターと、それぞれの説明を掲載します。

DFHJS2LS JCL プロシージャーは、データ・セット *HLQ*.XDFHINST にインストールされます。ここで、*HLQ* は、CICS がインストールされている高位修飾子です。

DFHJS2LS 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 (DFHJS2LS) を指定します。

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、DFHJS2LS で定義されます。

JAVADIR = *path*
DFHJS2LS 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/ に付加されて、/usr/lpp/*path* という完全なパス名が生成されます。

通常、このパラメーターは指定しません。 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。

PATHPREF = *prefix*
他のパラメーターで使用される z/OS UNIX ディレクトリー・パスを拡張する接頭部を指定します。接頭部を使用しない場合は、'' (空ストリング) を指定します。

通常、このパラメーターは指定しません。 デフォルト値は、**PATHPREF** 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。

SERVICE = *value*
このパラメーターは IBM サポートに指示された場合にのみ使用します。

TMPDIR = *tmpdir*
DFHJS2LS が一時ワークスペースとして使用する z/OS UNIX 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・ファイルの名前を構成するときに DFHJS2LS によって使用される接頭部を指定します。

デフォルト値は JS2LS です。

USSDIR = *path*
UNIX システム・サービスのファイル・システム内の 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/ に付加されて、/usr/lpp/cicsts/*path* という完全なパス名が生成されます。デフォルトを使用する場合は、これを '.' (ピリオド) として指定する必要があります。

通常、このパラメーターは指定しません。 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。

一時ワークスペース

DFHJS2LS 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。

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

/tmp/JS2LS.in

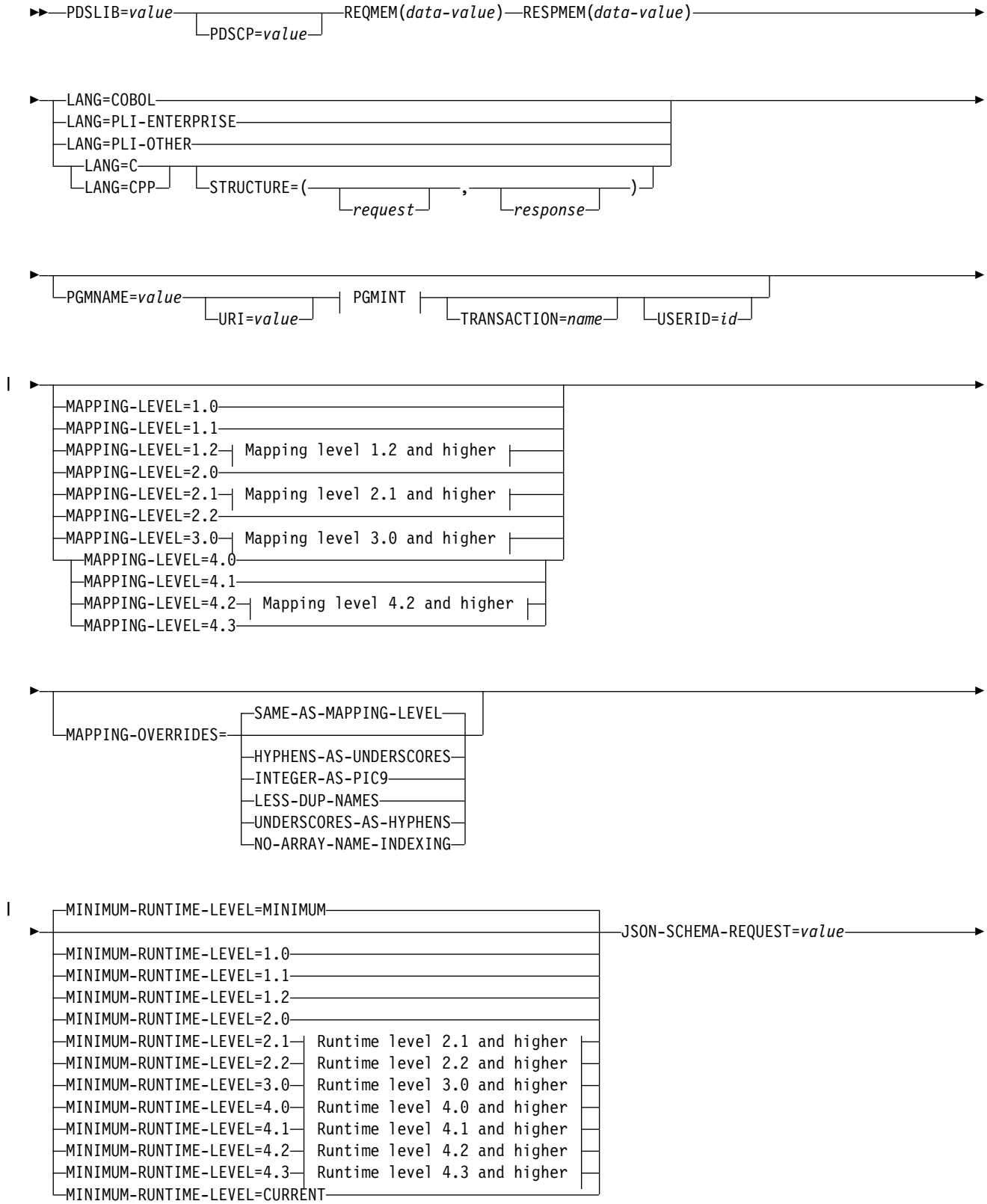
/tmp/JS2LS.out

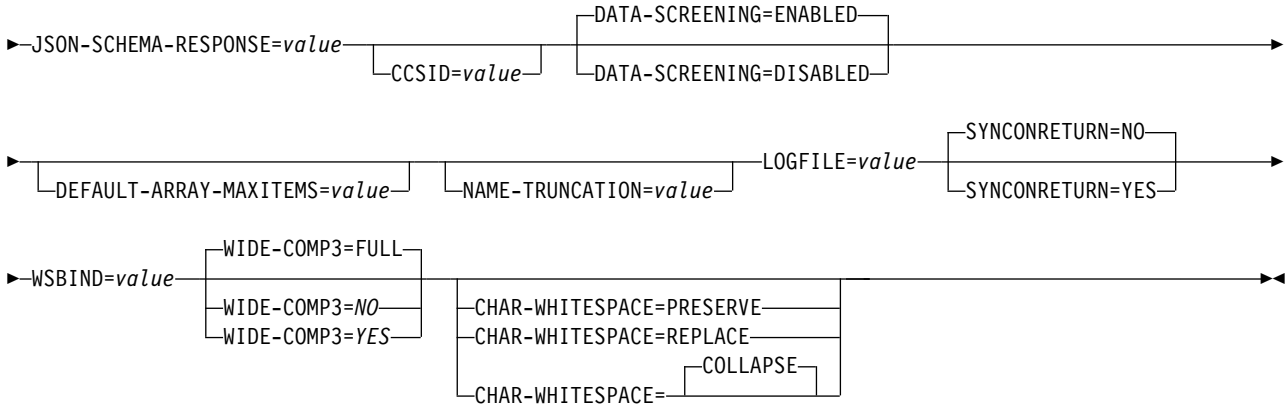
/tmp/JS2LS.err

重要: DFHJS2LS は、z/OS UNIX ファイルまたはデータ・セット・メンバーへのアクセスをロックしません。したがって、DFHJS2LS の複数のインスタンスが同時に実行され、同じ一時ワークスペース・ファイルを使用する場合、あるジョブがワークスペース・ファイルを使用しているときに、他のジョブがそれらのファイルを上書きするのを防止することができないため、予測不能な障害が生じます。

したがって、この状況を回避す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。これらの一時ファイルはジョブの終わりの前に削除されます。

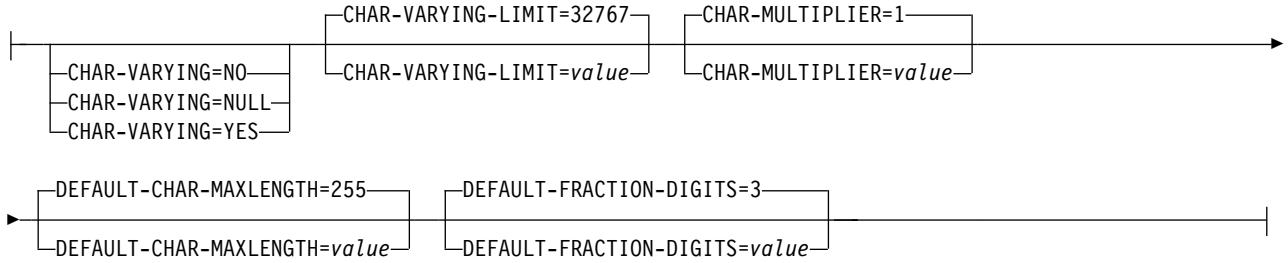
DFHJS2LS のパラメーターの入力





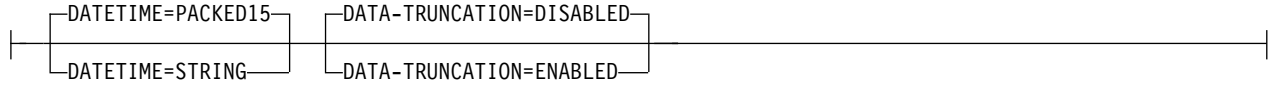
PGMINT:

マッピング・レベル **1.2** 以上の場合**:**

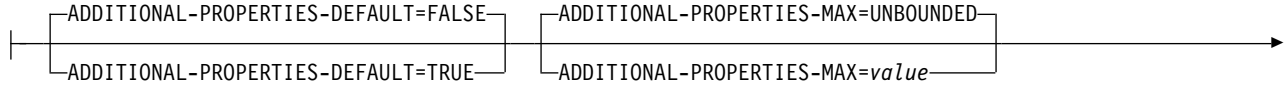


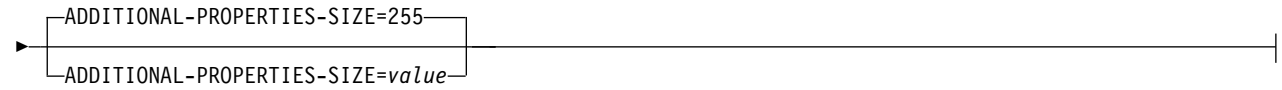
マッピング・レベル **2.1** 以上の場合**:**

マッピング・レベル **3.0** 以上**:**



マッピング・レベル **4.2** 以上の場合**:**





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。
- パラメーターおよび使用する場合は継続文字は、72 列を超えてはなりません。列 73 から 80 はブランクにする必要があります。
- 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。 スペースを含むアスタリスクより前の文字はすべてパラメーターの一部とみなされます。 以下に例を示します。

  ```
  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
  ```

- 行の先頭文字位置にある # 文字はコメント文字です。この行は無視されます。
- 行の末尾の文字の位置にコンマがある場合、これはオプションの行分離文字であり、無視されます。

パラメーターの記述

ADDITIONAL-PROPERTIES-DEFAULT = { **true** | **false** }

追加プロパティーのサポートを明示的に宣言していない JSON スキーマ・オブジェクトが、それらのプロパティーをサポートしていると解釈されるかどうかを示します。追加の JSON プロパティーは、JSON スキーマ内で事前に定義されていない JSON オブジェクト内のプロパティーです。これらのプロパティーは、通常、予期しない追加データとしてデータ変換メカニズムによって拒否されます。**ADDITIONAL-PROPERTIES-DEFAULT** が TRUE に設定されている場合、または JSON スキーマによってオブジェクトに対して additionalProperties:true が明示的に設定されている場合、生成されるコピーブックにこれらの値を保持するためのスペースが割り当てられます。アプリケーションは、コピーブック内の関連するフィールドを使用して、それらの値と対話できます。

このパラメーターは、マッピング・レベル 4.2 以上で使用できます。

ADDITIONAL-PROPERTIES-MAX = { **0-20** | **UNBOUNDED** }

追加プロパティーをサポートする JSON オブジェクトに対してサポートされるこれらのプロパティーの数を示します。**ADDITIONAL-PROPERTIES-DEFAULT** を参照してください。生成されるコピーブックには、追加プロパティーのアドレッシングに適した構造が含まれます。デフォルトでは、サポートされるプロパティーの数に最大制約はありません。コピーブックは、制約のない配列と同様の方法で生成され、コンテナーを使用します。このパラメーターを **INLINE-MAXOCCURS-LIMIT** パラメーターと組み合わせて使用可能な最大制約を適用し、最大数のプロパティー用に固定長の配列を割り当てることができます。これにより、コンテナーは不用になります。

このパラメーターは、マッピング・レベル 4.2 以上で使用できます。

ADDITIONAL-PROPERTIES-SIZE = { **16-32767** | **255** }

各 JSON 追加プロパティーの最大サイズを示します。JSON オブジェクトが **ADDITIONAL-PROPERTIES-DEFAULT** によって定義された追加プロパティーをサポートする場合、生成されるコピーブックには、**ADDITIONAL-PROPERTIES-MAX** で指定された数までプロパティーをサポートするバインディングが設定されます。デフォルトでは、各追加プロパティーでサポートされる最大値は 255 文字です。そのサイズのフィールドは、生成されるコピーブックに生成されます。このサイズは、**ADDITIONAL-PROPERTIES-SIZE** パラメーターを設定することでカスタマイズできます。例えば、JSON オブジェクトは、以下のプロパティーを含むように処理されます。

"example": { "notes": "this extra property was not defined in the JSON Schema" }

追加プロパティーをサポートするようにコピーブックが生成されている場合は、その値全体がアプリケーションに渡されて処理されます。この値は、プロパティーのキーの前の先頭引用符で始まり、プロパティーの値の末尾の右中括弧で終わります。この例では約 100 文字です。**ADDITIONAL-PROPERTIES-SIZE** に使用する値は、考えられる最大の値を保持できる大きさにする必要があります。処理される値に対して割り当てられたバッファーが小さすぎると、エラー応答が生成されます。

このパラメーターは、マッピング・レベル 4.2 以上で使用できます。

CCSID = *value*

アプリケーション・データ構造に文字データをエンコードするために、実行時に使用される CCSID を指定します。このパラメーターの値は、**LOCALCCSID** システム初期設定パラメーターの値を指定変更します。 *value* は、Java および z/OS Unicode Services ユーザーズ・ガイドおよび解説書でサポートされる EBCDIC CCSID である必要があります。このパラメーターを指定しない場合、アプリケーション・データ構造は、システム初期設定パラメーターで指定された CCSID を使用してエンコードされます。

CHAR-MULTIPLIER = { **1** | *value* }

マッピング・レベルが 1.2 以降のとき、各文字に許可されるバイト数を指定します。このパラメーターの *value* は、1 から 2,147,483,647 の範囲の正整数です。非数字に基づくマッピングはすべて、この乗数の対象です。バイナリー、数値、ゾーンおよびパック 10 進数フィールドは、この乗数の対象ではありません。

例えば、DBCS 文字を使用していて、実行時にすべての 2 バイト文字を潜在的なシフトアウトおよびシフトイン文字で囲むためのスペースを使用できるよう 3 の乗数を選択するときは、このパラメーターが便利です。

(UTF-16 を示す) **CCSID=1200** を設定する場合、**CHAR-MULTIPLIER** の有効値は 2 または 4 のみです。UTF-16 を使用する場合、有効値は 2 です。1 つの UTF-16 エンコード・ユニットを必要とする文字がアプリケーション・データに含まれると予期される場合は、**CHAR-MULTIPLIER=2** を使用してください。 2 つの UTF-16 エンコード・ユニットを必要とする文字がアプリケーション・データに含まれると予期される場合は、**CHAR-MULTIPLIER=4** を使用してください。

注: **CHAR-MULTIPLIER** を 1 に設定した場合、DBCS 文字は使用不可にはならず、2 に設定した場合、UTF-16 代理ペアは使用不可にはなりません。ただし、

ワイド文字が定期的に使用される場合、いくつかの有効値は、割り振られたフィールドに入らなくなります。より大きな **CHAR-MULTIPLIER** 値を使用すると、XML で有効な文字数よりも多くの文字を、割り振られたフィールドに格納できます。該当する範囲制限を守るように注意する必要があります。

CHAR-VARYING = { **NO** | **NULL** | **YES** }

マッピング・レベルが 1.2 以上のとき、可変長文字データがマップされる方法を指定します。 可変長バイナリー・データ・タイプは、常にコンテナーまたは可変構造のいずれかにマップされます。 このパラメーターを指定しない場合、デフォルトのマッピングは、指定された言語によって決まります。 以下のオプションを選択できます。

NO 可変長文字データは固定長ストリングとしてマップされます。

NULL
可変長文字データはヌル終了ストリングにマップされます。

YES 可変長文字データは PL/I では CHAR VARYING データ・タイプにマップされます。COBOL、C および C++ 言語では、可変長文字データは、2 つの関連エレメント (データ長およびデータ) から構成される同等の表現にマップされます。

CHAR-VARYING-LIMIT = { **32767** | *value* }

マッピング・レベルが 1.2 以上のとき、言語構造にマップされるバイナリー・データおよび可変長文字データの最大サイズを指定します。 文字データまたはバイナリー・データが、このパラメーターで指定される値より大きい場合は、コンテナーにマップされ、コンテナー名が生成された言語構造で使用されます。値の範囲は 0 からデフォルトの 32,767 バイトまでです。

CHAR-WHITESPACE = **COLLAPSE** | **REPLACE** | **PRESERVE**

ストリング型の値に含まれる空白を CICS でどのように処理するかを指定します。

COLLAPSE
先行空白、末尾空白、および組み込み空白が除去され、タブ、改行、および連続スペースはすべて単一のスペース文字に置き換えられます。

REPLACE
タブまたは改行が適切な数のスペースで置き換えられます。

PRESERVE
データ値に含まれる空白がすべて保持されます。

CHAR-WHITESPACE パラメーターが設定されていない場合、空白は省略されます。

注: このパラメーターは、空白が常に省略される date-time、uri、base64Binary、または hexBinary 形式のフィールドには適用されません。

CONTID = *value*

サービス・プロバイダーで、JSON メッセージを表すために使用される最上位のデータ構造を保持するコンテナーの名前を指定します。

CICS がターゲット・アプリケーション・プログラムに渡すコンテナーの長さは、要求コンテナーの長さおよび応答コンテナーの長さより大きくなります。

DATA-SCREENING = { **ENABLED** | **DISABLED** }
アプリケーション提供のデータのエラーを検査するかどうかを指定します。

ENABLED
言語構造と矛盾するアプリケーション提供のランタイム・データは、エラーとして扱われ、メッセージ DFHPI1010 が発行されます。エラー応答がアプリケーションに 返されます。

DISABLED
言語構造と矛盾するアプリケーション提供のランタイム・データの値は、 デフォルト値で置き換えられます。例えば、数値フィールド内のゼロは誤った値を置き換えます。メッセージ DFHPI1010 は発行されず、通常の応答がアプリケーションに返されます。この機能を使用して、 初期設定されていない出力フィールドから生成される INVALID_PACKED_DEC および INVALID_ZONED_DEC エラー応答を回避できます。

DATA-TRUNCATION = { **DISABLED** | **ENABLED** }
固定長フィールド構造で可変長データを許容するかどうかを指定します。

DISABLED
データが CICS が予期する固定長より短い場合に、CICS は切り捨てられたデータを拒否してエラー・メッセージを発行します。

ENABLED
データが CICS が予期する固定長より短い場合に、CICS は切り捨てられたデータを許容して、欠落データをヌル値として処理します。

DATETIME = { **PACKED15** | **STRING** }
JSON 日時エレメントを言語構造にマップする方法を指定します。

PACKED15
デフォルトでは、すべての JSON 日時エレメントがタイム・スタンプとして処理され、CICS ABSTIME 形式にマップされます。

STRING
JSON 日時エレメントはテキストとして処理されます。

DEFAULT-ARRAY-MAXITEMS = *value*
JSON スキーマで最大出現回数の情報 (maxItems) が示されていない場合に適用する最大配列境界を指定します。このパラメーターが設定されていない場合、最大制限は適用されません。このパラメーターの値は、1 から 2147483647 までの範囲の正整数です。このパラメーターを **INLINE-MAXOCCURS-LIMIT** パラメーターとともに使用することによって、 JSON 配列が言語構造にマップされる方法を制御することができます。

DEFAULT-CHAR-MAXLENGTH = { **255** | *value* }
マッピング・レベルが 1.2 以上のとき、Web サービス記述文書に長さが暗黙指定されていない場合に、マッピングについて文字データのデフォルトの配列長を文字数で指定します。 このパラメーターの value は、1 から 2,147,483,647 の範囲の正整数です。

DEFAULT-FRACTION-DIGITS = { **3** | *value* }
JSON 10 進スキーマ・タイプで使用するデフォルトの小数桁数を指定します。デフォルト値は 3 です。 COBOL の場合、有効な範囲は 0 から 17、または

パラメーター **WIDE-COMP3** が使用されている場合、0 から 30 です。 C または PLI の場合、有効な範囲は 0 から 30 までです。

INLINE-MAXOCCURS-LIMIT = { **1** | *value* }
インラインの可変反復内容を maxItems JSON スキーマ・キーワードに基づいて使用するかどうかを指定します。インラインにマップされる可変の繰り返しコンテンツは、生成される言語構造の残りの部分と共に現在のコンテナーに入ります。 可変の繰り返しコンテンツは、2 つの部分 (データの出現回数を格納するカウンターと、データのそれぞれの出現を格納する配列) に分けて格納されます。 可変繰り返しコンテンツに代わるマッピングとして、コンテナー・ベースのマッピングがあります。この場合、データの出現回数、およびデータを収容するコンテナーの名前を格納します。別個のコンテナーにデータを格納するとパフォーマンスに影響を与えるため、インライン・マッピングの方が適しているかもしれません。

INLINE-MAXOCCURS-LIMIT パラメーターは、マッピング・レベル 2.1 以降でのみ使用できます。 **INLINE-MAXOCCURS-LIMIT** の値は、 0 から 32,767 の範囲の正整数です。 値 0 は、インライン・マッピングを使用しないことを示します。値 1 を使用すると、オプション・エレメントがインラインでマップされます。maxOccurs 属性の *value* が **INLINE-MAXOCCURS-LIMIT** の *value* より大きい場合、コンテナーに基づくマッピングが使用されます。それ以外の場合はインライン・マッピングが使用されます。

可変の繰り返しリストをインラインでマップするかどうか決定する際には、繰り返されるデータの単一項目の長さを考慮してください。 長い項目が少ない回数だけ出現する場合は、コンテナーに基づくマッピングが適しています。短い項目が数多く出現する場合は、インライン・マッピングが適しています。

JSON-SCHEMA-REQUEST = *value*
これは必須パラメーターです。

この値は、要求 JSON スキーマが格納される UNIX システム・サービスの場所を示します。

JSON-SCHEMA-RESPONSE = *value*
これは必須パラメーターです。

この値は、応答 JSON スキーマが格納される UNIX システム・サービスの場所を示します。

LANG = **COBOL**|**PLI-ENTERPRISE**|**PLI-OTHER**|**C**|**CPP**
高水準言語構造のプログラミング言語を指定します。

COBOL
COBOL

PLI-ENTERPRISE
Enterprise PL/I

PLI-OTHER
Enterprise PL/I 以外の PL/I のレベル

C C

CPP C++

LOGFILE = *value*

DFHJS2LS 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 z/OS UNIX 名です。 DFHJS2LS は、このファイルが存在しない場合、ディレクトリー構造は作成しませんがファイルを作成します。

通常はこのファイルを使用しませんが、DFHJS2LS 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。

MAPPING-LEVEL = **{ 1.0 | 1.1 | 1.2 | 2.0 | 2.1 | 2.2 | 3.0 | 4.0 | 4.1 | 4.2 | 4.3 }**

Web サービス・バインディング・ファイルおよび言語構造を生成する際に、DFHJS2LS が使用するマッピングのレベルを指定します。 以下のオプションを選択できます。

1.0 このオプションは、SOAP Web サービスとの互換性のために保持されています。 JSON で使用することは推奨されていません。

1.1 このオプションは、SOAP Web サービスとの互換性のために保持されています。 JSON で使用することは推奨されていません。

1.2 このオプションは、SOAP Web サービスとの互換性のために保持されています。 JSON で使用することは推奨されていません。

2.0 このオプションは、SOAP Web サービスとの互換性のために保持されています。 JSON で使用することは推奨されていません。

2.1 このオプションは、SOAP Web サービスとの互換性のために保持されています。 JSON で使用することは推奨されていません。

2.2 このオプションは、SOAP Web サービスとの互換性のために保持されています。 JSON で使用することは推奨されていません。

3.0 このマッピング・レベルは、使用可能なすべてのオプション・セットを使用して JSON スキーマを生成する場合に使用します。

4.0 UTF-16 を使用する場合には、CICS TS 5.2 以降の領域でこのマッピング・レベルを使用します。

4.1 切り捨て可能な配列をサポートするには、CICS TS 5.2 以降の領域でこのマッピング・レベルを使用します。

4.2 追加プロパティーについては、CICS TS 5.4 以降の領域でこのマッピング・レベルを使用します。

4.3 多次元配列をサポートするには、CICS TS 5.4 以降の領域でこのマッピング・レベルを使用します。

マッピング・レベルについて詳しくは、CICS アシスタントのマッピング・レベルを参照してください。

MAPPING-OVERRIDES = **{ SAME-AS-MAPPING-LEVEL |HYPHENS-AS-UNDERSCORES | INTEGER-AS-PIC9 | LESS-DUP-NAMES | UNDERSCORES-AS-HYPHENS | NO-ARRAY-NAME-INDEXING }**

言語構造を生成するときの指定されたマッピング・レベルについて、デフォルトの動作を指定変更するかどうかを指定します。

SAME-AS-MAPPING-LEVEL
このパラメーターは、マッピング・レベルと同じスタイルで言語構造を生成します。これはデフォルトです。

HYPHENS-AS-UNDERSCORES
PL/I のみ。このパラメーターは、生成される PL/I 言語構造を読みやすくするために、JSON スキーマ内のすべてのハイフンを文字 X ではなく下線に変換します。詳しくは、JSON スキーマから PL/I へのマッピングを参照してください。このオプションは、マッピング・レベル 4.2 で自動的に有効になります。

INTEGER-AS-PIC9
COBOL および DFHJS2LS のみ。このパラメーターは、JSON スキーマからの整数値を、英数字ではなく数字として含む言語構造を生成します。このオプションは、マッピング・レベル 4.0 で自動的に有効になります。

LESS-DUP-NAMES
このパラメーターは、フィールドを直接参照できるように名前の末尾に _value を付けた非構造性の構造フィールド名を生成します。例えば、以下の PL/I 言語構造で、MAPPING-OVERRIDES=LESS-DUP-NAMES が指定されるとき、レベル 12 フィールドの streetName には接尾部に _value が付いています。

```
09 streetName,
12 streetName CHAR(255) VARYING
UNALIGNED,
12 filler BIT (7),
12 attr_nil_streetName_value BIT (1),
```

結果構造は以下のとおりです。

```
09 streetName,
12 streetName_value CHAR(255) VARYING
UNALIGNED,
12 filler BIT (7),
12 attr_nil_streetName_value BIT (1),
```

このオプションは、マッピング・レベル 4.2 で自動的に有効になります。

UNDERSCORES-AS-HYPHENS
COBOL のみ。 このパラメーターは、生成される COBOL 言語構造を読みやすくするために、JSON スキーマ内のすべての下線を文字 X ではなくハイフンに変換します。フィールド名の競合が発生する場合、そのフィールドが必ず固有になるように番号が付きます。 詳しくは、JSON スキーマから COBOL へのマッピングを参照してください。

このオプションは、マッピング・レベル 4.0 で自動的に有効になります。

NO-ARRAY-NAME-INDEXING
COBOL および Enterprise PL/I のみ。配列内のフィールド名が上位構造のスコープ内でのみ固有であるようにします。

MINIMUM-RUNTIME-LEVEL = { MINIMUM | 1.0 | 1.1 | 1.2 | 2.0 | 2.1 | 2.2 | 3.0 | 4.0 | 4.1 | 4.2 | 4.3 | CURRENT }

Web サービス・バインディング・ファイルを配置できる最小の CICS 実行時環境を指定します。指定した他のパラメーターと一致しないレベルを選択すると、エラー・メッセージが送信されます。 以下のオプションを選択できます。

MINIMUM
選択したパラメーターを前提として、最低限可能な CICS 実行時レベルが自動的に割り振られます。

1.0 このオプションは、SOAP Web サービスとの互換性のために保持されています。 JSON で使用することは推奨されていません。

1.1 このオプションは、SOAP Web サービスとの互換性のために保持されています。 JSON で使用することは推奨されていません。

1.2 このオプションは、SOAP Web サービスとの互換性のために保持されています。 JSON で使用することは推奨されていません。

2.0 このオプションは、SOAP Web サービスとの互換性のために保持されています。 JSON で使用することは推奨されていません。

2.1 このオプションは、SOAP Web サービスとの互換性のために保持されています。 JSON で使用することは推奨されていません。

2.2 このオプションは、SOAP Web サービスとの互換性のために保持されています。 JSON で使用することは推奨されていません。

3.0 生成された Web サービス・バインディング・ファイルは、CICS TS 4.1 以降の領域にデプロイされます。

注: JSON サポートが使用可能なのは、CICS TS 4.2 以降に限られます。

4.0 生成された Web サービス・バインディング・ファイルは、CICS TS 5.2 以降の領域に正常にデプロイされます。この実行時レベルでは、**MAPPING-LEVEL** パラメーターにマッピング・レベル 4.0 以前を使用できます。このレベルでは任意のオプション・パラメーターを使用できます。

4.1 生成された Web サービス・バインディング・ファイルは、CICS TS 5.2 以降の領域に正常にデプロイされます。この実行時レベルでは、**MAPPING-LEVEL** パラメーターには、マッピング・レベル 4.1 以前を使用することができます。

4.2 生成された Web サービス・バインディング・ファイルは、CICS TS 5.4 以降の領域に正常にデプロイされます。この実行時レベルでは、**MAPPING-LEVEL** パラメーターには、マッピング・レベル 4.2 以前を使用することができます。

4.3 生成された Web サービス・バインディング・ファイルは、CICS TS 5.4 以降の領域に正常にデプロイされます。この実行時レベルでは、**MAPPING-LEVEL** パラメーターには、マッピング・レベル 4.3 以前を使用することができます。

CURRENT
生成された Web サービス・バインディング・ファイルは、Web サー

ビス・バインディング・ファイルの生成に使用するものと同じ実行時レベルで、CICS 領域に正常に配置されます。

NAME-TRUNCATION = **{ LEFT | RIGHT }**
JSON 名が左と右のどちらから切り捨てられるかを指定します。 CICS Web サービス・アシスタントは、指定された高水準言語における適切な長さに JSON 名を切り捨てます。デフォルトでは、名前は右から切り捨てられます。

PDSCP = *value*
REQMEM および **RESPMEM** パラメーターに指定される区分データ・セット・メンバーで使用されるコード・ページを指定します。ここで、*value* は CCSID 番号または Java コード・ページ番号です。このパラメーターを指定しない場合は、z/OS UNIX システム・サービスのコード・ページが使用されます。 例えば、**PDSCP**=037 と指定できます。

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、**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。

ターゲット・アプリケーション・プログラムが要求を処理するとき、同じメカニズムを使用して応答を戻す必要があります。要求が通信域で受信されている場合は、応答を通信域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、応答をコンテナーに戻す必要があります。CICS がターゲット・アプリケーション・プログラムに渡す通信域またはコンテナーの長さは、要求の通信域またはコンテナーの長さ、および応答の通信域またはコンテナーの長さより大きくなります。

PGMNAME = *value*
CICS PROGRAM リソース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 DFHJS2LS 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・プログラムのリソース名を指定します。

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 DFHJS2LS を使用する場合、このパラメーターを省略します。

REQMEM = *value*
アプリケーション・プログラムへの入力データとなる Web サービス要求の高

水準言語構造が格納されている区分データ・セット・メンバーの名前を生成するときに DFHJS2LS が使用する 1 から 6 文字の接頭部を指定します。

DFHJS2LS は接頭部に 01 を付加することにより、区分データ・セット・メンバーの名前を生成します。

RESPMEM = *value*
アプリケーション・プログラムからの出力データとなる Web サービス応答の高水準言語構造が格納されている区分データ・セット・メンバーの名前を生成するときに DFHJS2LS が使用する 1 から 6 文字の接頭部を指定します。

DFHJS2LS は接頭部に 01 を付加することにより、区分データ・セット・メンバーの名前を生成します。

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。

生成される要求構造と応答構造には、*request01* および *response01* という名前が付きます。
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。

SYNCONRETURN = { **NO** | **YES** }
リモート Web サービスが同期点を発行できるかどうかを指定します。

NO リモート Web サービスは同期点を発行できません。 この値はデフォルトです。リモート Web サービスが同期点を発行すると、ADPL 異常終了になり失敗します。

YES リモート Web サービスは同期点を発行できます。 YES を選択した場合、リモート Web サービスから制御が戻されたときにリモート・タスクが別個の作業単位としてコミットされます。リモート Web サービスがリカバリー可能リソースを更新して戻した後に障害が発生した場合、そのリソースの更新内容をバックアウトすることはできません。

TRANSACTION = *name*
サービス・プロバイダーで、このパラメーターは、応答を組み立てるためにパイプラインを開始できる 1 から 4 文字の別名トランザクションの名前を指定します。 このパラメーターの値は、URIMAP リソースが **PIPELINE** スキャン・コマンドを使用して自動的に作成される際に、URIMAP リソースの TRANSACTION 属性を定義するために使用されます。

許容文字:
A-Z a-z 0-9 $ @ # _ < >

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相対 URI を指定します。 CICS は、DFHJS2LS 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 のパスのコンポーネントを指定します。

USERID = *id*

サービス・プロバイダーでは、このパラメーターは、任意の Web クライアントで使用可能な 1 文字から 8 文字のユーザー ID を指定します。 アプリケーションから生成される応答や Web サービスについては、そのユーザー ID の下で別名トランザクションが追加されます。 このパラメーターの値は、URIMAP リソースが **PIPELINE** スキャン・コマンドを使用して自動的に作成される際に、URIMAP リソースの USERID 属性を定義するために使用されます。

許容文字:
A-Z a-z 0-9 $ @ #

WIDE-COMP3 = { **FULL** | **NO** | **YES** }

生成される COBOL または PL/I 言語構造でパック 10 進数の可変長の最大サイズを制御します。

FULL COBOL および PL/I の場合。DFHJS2LS は、すべての有効な値を保持できるサイズでパック 10 進数フィールドを生成します。最大サイズは 31 桁です。これはデフォルトです。

NO COBOL のみ。 DFHJS2LS は、COBOL 言語構造タイプ COMP-3 を生成しているとき、パック 10 進数の可変長を 18 に制限します。 パック 10 進数のサイズが 18 より大きい場合、指定されたタイプは合計 18 桁に制限されていることを示すメッセージ DFHPI9022W が出されます。

YES COBOL のみ。 DFHJS2LS は、COBOL 言語構造タイプ COMP-3 を生成しているとき、最大サイズ 31 をサポートします。

注: NO および YES オプションを指定すると、すべての有効な値を表わすことができないフィールドが生成されます。FULL オプションは、この問題を回避します。ただし、FULL オプションを指定すると、パック 10 進数フィールドで無効な値が表される可能性があります。例えば、スキーマに最大サイズとして最大 5 桁の数字と最大 2 桁の小数点以下の数値が指定されている場合、FULL オプションを指定すると、7 桁の数字が許容されるパック 10 進数フィールドが生成されます。この場合、25000 や 999.99 などの有効な値を収容するだけのスペースがあると同時に、9999.99 などの無効な値を収容するだけのスペースも提供されることになります。FULL オプションを使用する場合は、アプリケーション・データに無効な値が生成されないよう注意してください。

WSBIND = *value*

Web サービス・バインディング・ファイルの完全修飾 z/OS UNIX 名です。DFHJS2LS は、このファイルが存在しない場合、ディレクトリー構造は作成しませんがファイルを作成します。デフォルトのファイル拡張子は .wsbind です。

その他の情報

- DFHJS2LS を実行するユーザー ID は、UNIX システム・サービスを使用するように構成されていなければなりません。 ユーザー ID には、CICS z/OS UNIX ファイル構造および PDS ライブラリーに対する読み取り権限、および **LOGFILE**、**WSBIND**、および **WSDL** パラメーターで指定されたディレクトリーへの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。
- JCL の最大パラメーター長は 100 文字です。 **STDPARM** ステートメントを使用してこの値を増やすことができます。詳細については、z/OS UNIX システム・サービス ユーザーズ・ガイドを参照してください。

例

```
//JS2LS JOB '
accounting information
',
name,MSGCLASS=A
// SET QT=''''
//JAVAPROG EXEC DFHJS2LS,
// TMPFILE=&QT.&SYSUID.&QT
/INPUT.SYSUT1 DD *
PDSLIB=//CICSHLQ.SDFHSAMP
REQMEM=CPYBK1
RESPMEM=CPYBK2
JSON-SCHEMA-REQUEST=example.json
JSON-SCHEMA-RESPONSE=example.json
LANG=COBOL
LOGFILE=/u/exampleapp/wsbind/example.log
MAPPING-LEVEL=4.0
CHAR-VARYING=NULL
INLINE-MAXOCCURS-LIMIT=2
PGMNAME=DFH0XCMN
URI=exampleApp/example
PGMINT=COMMAREA
SYNCONRETURN=YES
WSBIND=/u/exampleapp/wsbind/example.wsbind
/*
```

DFHJS2LS: RESTful サービスにおける JSON スキーマから高水準言語への変換

DFHJS2LS プロシージャーは JSON スキーマから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。 DFHJS2LS は、RESTful JSON サービス・プロバイダーを作成するときに準備段階で使用できます。このトピックでは、DFHJS2LS のジョブ制御ステートメント、シンボリック・パラメーター、入力パラメーターおよびそれぞれの説明を記載します。

DFHJS2LS JCL プロシージャーは、データ・セット *HLQ*.XDFHINST にインストールされます。ここで、*HLQ* は CICS がインストールされている高位修飾子です。

DFHJS2LS 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 (DFHJS2LS) を指定します。

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、DFHJS2LS で定義されます。

JAVADIR = *path*
DFHJS2LS 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/ に付加されて、/usr/lpp/*path* という完全なパス名が生成されます。

通常、このパラメーターは指定しません。 デフォルト値は、**JAVADIR** パラメーターで CICS インストール・ジョブ (DFHISTAR) に指定された値です。

PATHPREF = *prefix*
他のパラメーターで使用される z/OS UNIX ディレクトリー・パスを拡張する接頭部を指定します。接頭部を使用しない場合は、'' (空ストリング) を指定します。

通常、このパラメーターは指定しません。 デフォルト値は、**PATHPREF** 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。

SERVICE = *value*
このパラメーターは IBM サポートに指示された場合にのみ使用します。

TMPDIR = *tmpdir*
DFHJS2LS が一時ワークスペースとして使用する z/OS UNIX 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・ファイルの名前を構成するときに DFHJS2LS によって使用される接頭部を指定します。

デフォルト値は JS2LS です。

USSDIR = *path*
UNIX システム・サービスのファイル・システム内の 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/ に付加されて、/usr/lpp/cicsts/*path* という完全なパス名が生成されます。デフォルトを使用する場合は、これを '.' (ピリオド) として指定する必要があります。

通常、このパラメーターは指定しません。 デフォルト値は、**USSDIR** パラメーターで CICS インストール・ジョブ (DFHISTAR) に指定された値です。

一時ワークスペース

DFHJS2LS 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

各部の意味は次のとおりです。

tmpdir **TMPDIR** パラメーターで指定された値です。

tmpprefix **TMPFILE** パラメーターで指定された値です。

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

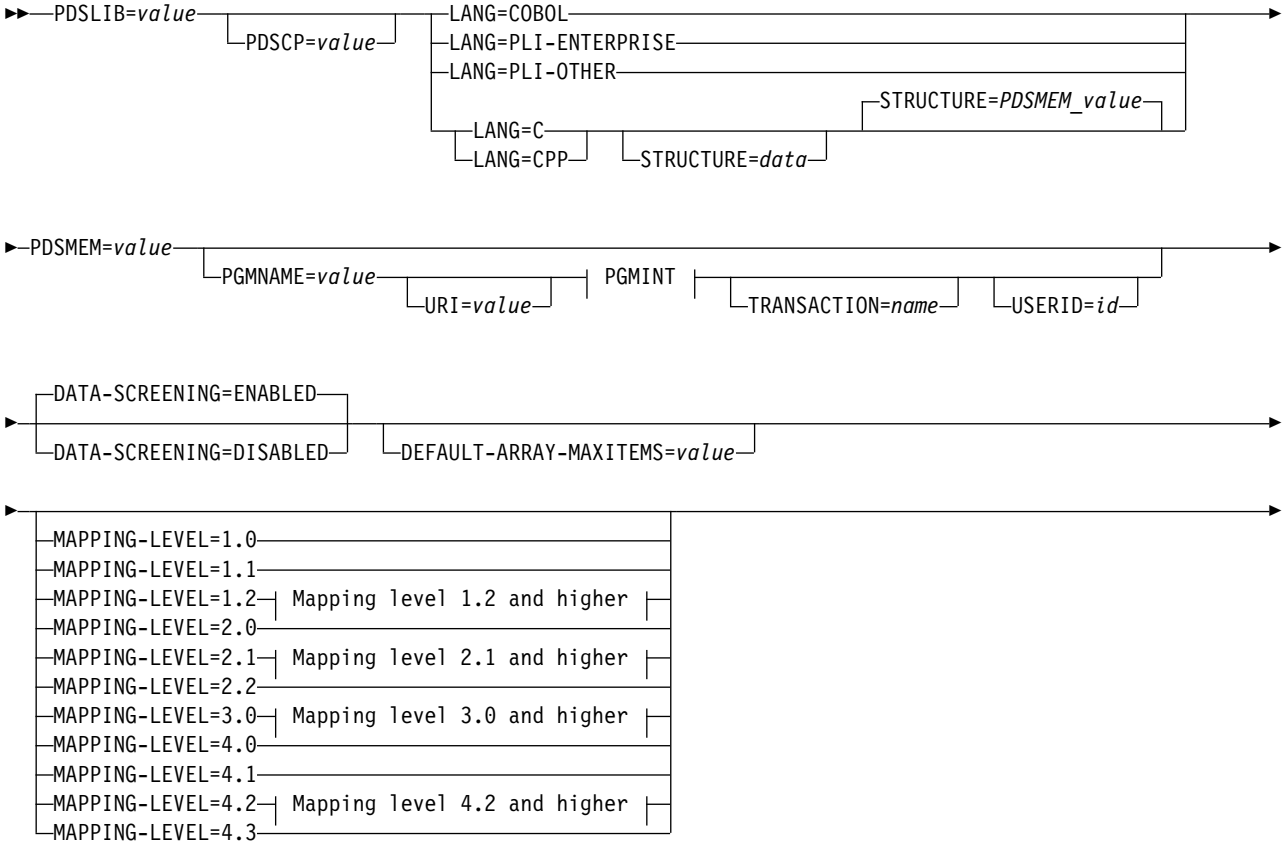
/tmp/JS2LS.in

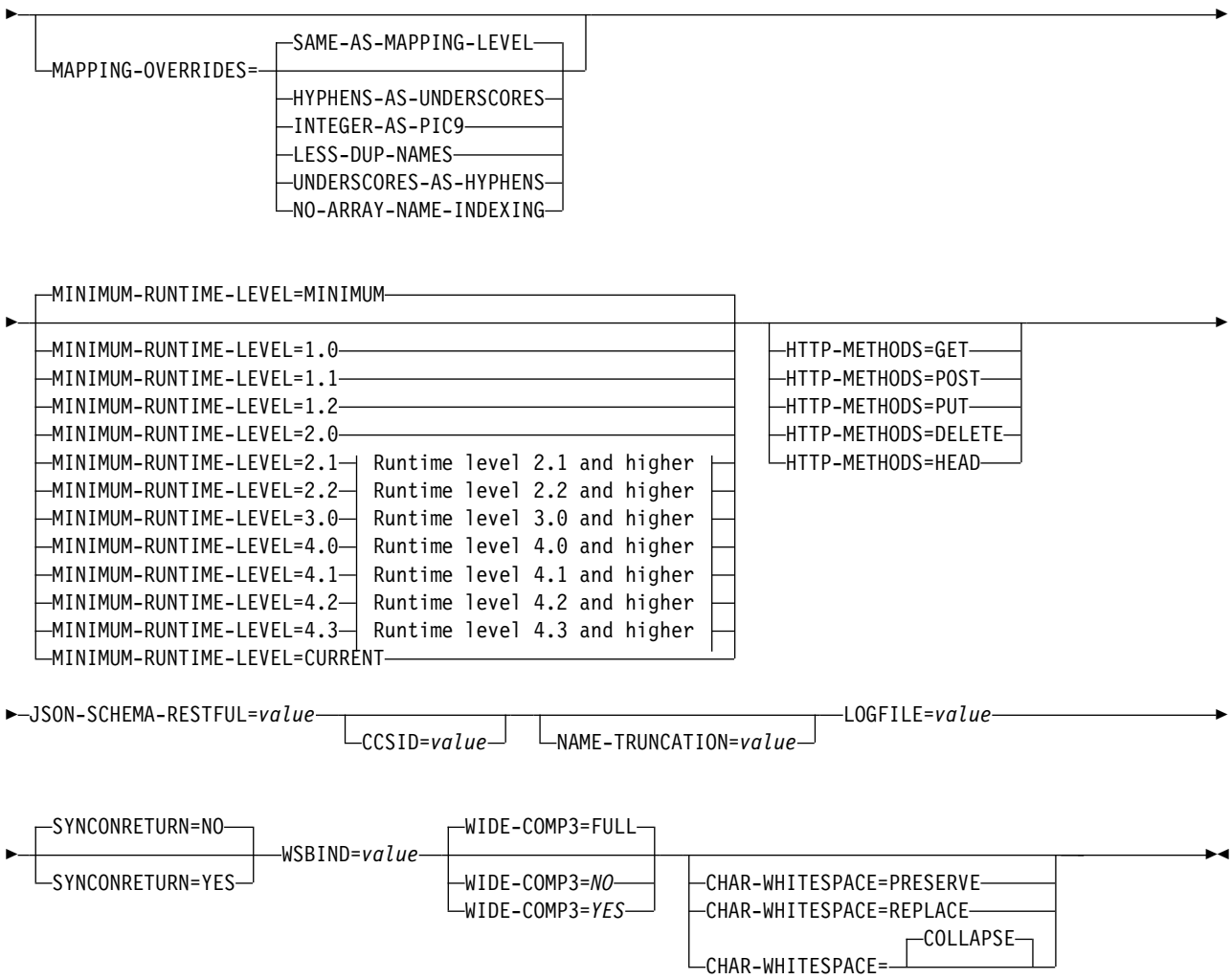
```
/tmp/JS2LS.out
/tmp/JS2LS.err
```

重要: DFHJS2LS は、z/OS UNIX ファイルまたはデータ・セット・メンバーへのアクセスをロックしません。したがって、DFHJS2LS の複数のインスタンスが同時に実行され、同じ一時ワークスペース・ファイルを使用する場合、あるジョブがワークスペース・ファイルを使用しているときに、他のジョブがそれらのファイルを上書きするのを防止することができないため、予測不能な障害が生じます。

したがって、この状況を回避す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。これらの一時ファイルはジョブの終わりの前に削除されます。

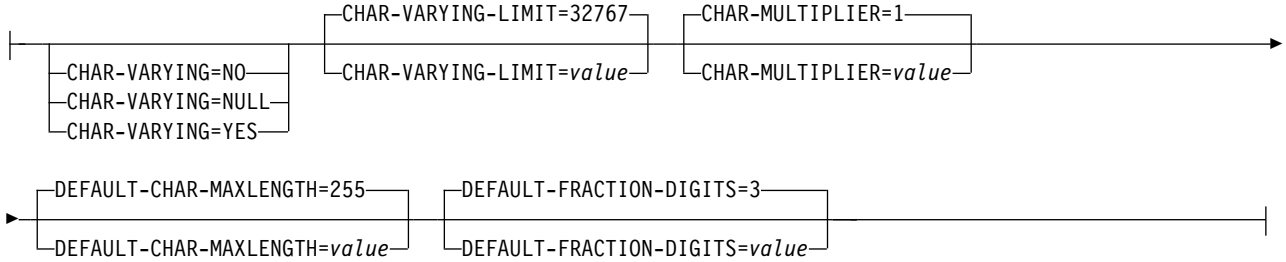
DFHJS2LS のパラメーターの入力



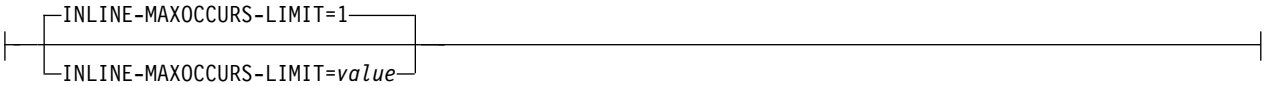


PGMINT:

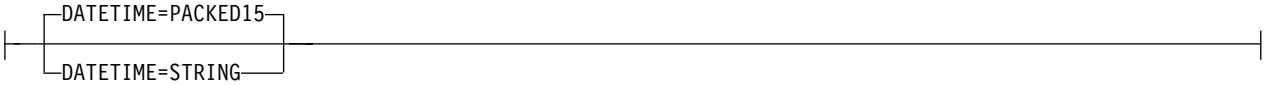
マッピング・レベル **1.2** 以上の場合:



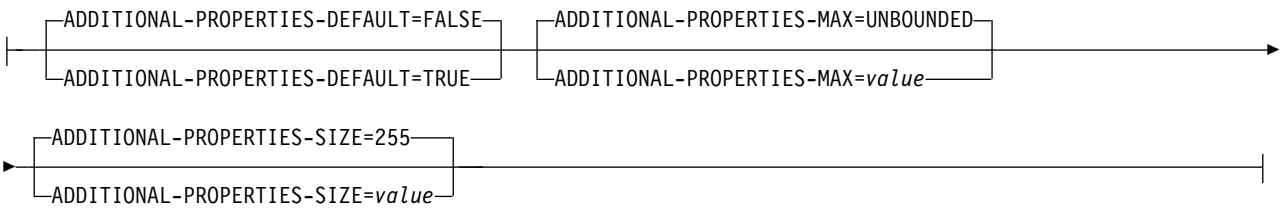
マッピング・レベル **2.1** 以上の場合:



マッピング・レベル **3.0** 以上:



マッピング・レベル **4.2** 以上の場合:



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。
- パラメーターおよび使用する場合は継続文字は、72 列を超えてはなりません。列 73 から 80 はブランクにする必要があります。
- 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。 スペースを含むアスタリスクより前の文字はすべてパラメーターの一部とみなされます。 以下に例を示します。

  ```
  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。
- 行の末尾の文字の位置にコンマがある場合、これはオプションの行分離文字であり、無視されます。

パラメーターの記述

ADDITIONAL-PROPERTIES-DEFAULT = { **true** | **false** }

追加プロパティーのサポートを明示的に宣言していない JSON スキーマ・オブジェクトが、それらのプロパティーをサポートしていると解釈されるかどうかを示します。追加の JSON プロパティーは、JSON スキーマ内で事前に定義されていない JSON オブジェクト内のプロパティーです。これらのプロパティーは、通常、予期しない追加データとしてデータ変換メカニズムによって拒否され

ます。**ADDITIONAL-PROPERTIES-DEFAULT** が TRUE に設定されている場合、または JSON スキーマによってオブジェクトに対して additionalProperties:true が明示的に設定されている場合、生成されるコピーブックにこれらの値を保持するためのスペースが割り当てられます。アプリケーションは、コピーブック内の関連するフィールドを使用して、それらの値と対話できます。

このパラメーターは、マッピング・レベル 4.2 以上で使用できます。

ADDITIONAL-PROPERTIES-MAX = { **0-20** | **UNBOUNDED** }
追加プロパティーをサポートする JSON オブジェクトに対してサポートされるこれらのプロパティーの数を示します。**ADDITIONAL-PROPERTIES-DEFAULT** を参照してください。生成されるコピーブックには、追加プロパティーのアドレッシングに適した構造が含まれます。デフォルトでは、サポートされるプロパティーの数に最大制約はありません。コピーブックは、制約のない配列と同様の方法で生成され、コンテナーを使用します。このパラメーターを **INLINE-MAXOCCURS-LIMIT** パラメーターと組み合わせて使用可能な最大制約を適用し、最大数のプロパティー用に固定長の配列を割り当てることができます。これにより、コンテナーは不用になります。

このパラメーターは、マッピング・レベル 4.2 以上で使用できます。

ADDITIONAL-PROPERTIES-SIZE = { **16-32767** | **255** }
各 JSON 追加プロパティーの最大サイズを示します。JSON オブジェクトが **ADDITIONAL-PROPERTIES-DEFAULT** によって定義された追加プロパティーをサポートする場合、生成されるコピーブックには、**ADDITIONAL-PROPERTIES-MAX** で指定された数までプロパティーをサポートするバインディングが設定されます。デフォルトでは、各追加プロパティーでサポートされる最大値は 255 文字です。そのサイズのフィールドは、生成されるコピーブックに生成されます。このサイズは、**ADDITIONAL-PROPERTIES-SIZE** パラメーターを設定することでカスタマイズできます。例えば、JSON オブジェクトは、以下のプロパティーを含むように処理されます。

"example": { "notes": "this extra property was not defined in the JSON Schema" }

追加プロパティーをサポートするようにコピーブックが生成されている場合は、その値全体がアプリケーションに渡されて処理されます。この値は、プロパティーのキーの前の先頭引用符で始まり、プロパティーの値の末尾の右中括弧で終わります。この例では約 100 文字です。**ADDITIONAL-PROPERTIES-SIZE** に使用する値は、考えられる最大の値を保持できる大きさにする必要があります。処理される値に対して割り当てられたバッファーが小さすぎると、エラー応答が生成されます。

このパラメーターは、マッピング・レベル 4.2 以上で使用できます。

CCSID = *value*
アプリケーション・データ構造に文字データをエンコードするために、実行時に使用される CCSID を指定します。このパラメーターの値は、**LOCALCCSID** システム初期設定パラメーターの値を指定変更します。 *value* は、Java および z/OS Unicode Services ユーザーズ・ガイドおよび解説書でサポートされる EBCDIC CCSID である必要があります。このパラメーターを指定しない場合、アプリケーション・データ構造は、システム初期設定パラメーターで指定された CCSID を使用してエンコードされます。

CHAR-MULTIPLIER = { **1** | *value* }

マッピング・レベルが 1.2 以降のとき、各文字に許可されるバイト数を指定します。このパラメーターの *value* は、1 から 2,147,483,647 の範囲の正整数です。 非数字に基づくマッピングはすべて、この乗数の対象です。バイナリー、数値、ゾーンおよびパック 10 進数フィールドは、この乗数の対象ではありません。

このパラメーターが役に立つのは、例えば、実行時にすべての 2 バイト文字の周囲に潜在的なシフトアウト文字とシフトイン文字のスペースを許可するために、乗数 3 を選択できる DBCS 文字の使用を計画している場合などです。

(UTF-16 を示す) **CCSID=1200** を設定する場合、**CHAR-MULTIPLIER** の有効値は 2 または 4 のみです。 UTF-16 を使用する場合のデフォルト値は 2 です。1 つの UTF-16 エンコード・ユニットを必要とする文字がアプリケーション・データに含まれると予期される場合は、**CHAR-MULTIPLIER=2** を使用してください。 2 つの UTF-16 エンコード・ユニットを必要とする文字がアプリケーション・データに含まれると予期される場合は、**CHAR-MULTIPLIER=4** を使用してください。

注: **CHAR-MULTIPLIER** を 1 に設定した場合、DBCS 文字は使用不可にはならず、2 に設定した場合、UTF-16 代理ペアは使用不可にはなりません。ただし、ワイド文字が定期的に使用される場合、いくつかの有効値は、割り振られたフィールドに入らなくなります。より大きな **CHAR-MULTIPLIER** 値を使用すると、XML で有効な文字数よりも多くの文字を、割り振られたフィールドに格納できます。該当する範囲制限を守るように注意する必要があります。

CHAR-VARYING = **NO** | **NULL** | **YES**

マッピング・レベルが 1.2 以上のとき、可変長文字データがマップされる方法を指定します。 可変長バイナリー・データ・タイプは、常にコンテナーまたは可変構造のいずれかにマップされます。 このパラメーターを指定しない場合は、指定される言語に応じてデフォルトのマッピングが異なります。 以下のオプションを選択できます。

NO 可変長文字データは固定長ストリングとしてマップされます。

NULL
可変長文字データはヌル終了ストリングにマップされます。

YES 可変長文字データは PL/I では CHAR VARYING データ・タイプにマップされます。COBOL、C および C++ 言語では、可変長文字データは、2 つの関連エレメント (データ長およびデータ) から構成される同等の表現にマップされます。

CHAR-VARYING-LIMIT = **32767** | *value*

マッピング・レベルが 1.2 以上のとき、言語構造にマップされるバイナリー・データおよび可変長文字データの最大サイズを指定します。 文字データまたはバイナリー・データが、このパラメーターで指定される値より大きい場合は、コンテナーにマップされ、コンテナー名が生成された言語構造で使用されます。値の範囲は 0 からデフォルトの 32,767 バイトまでです。

CHAR-WHITESPACE = **COLLAPSE** | **REPLACE** | **PRESERVE**

ストリング型の値に含まれる空白を CICS でどのように処理するかを指定します。

COLLAPSE
先行空白、末尾空白、および組み込み空白が除去され、タブ、改行、および連続スペースはすべて単一のスペース文字に置き換えられます。

REPLACE
タブまたは改行が適切な数のスペースで置き換えられます。

PRESERVE
データ値に含まれる空白がすべて保持されます。

CHAR-WHITESPACE パラメーターを設定しないと、空白が省略されます。

注: このパラメーターは、空白が常に省略される date-time、uri、base64Binary、または hexBinary 形式のフィールドには適用されません。

CONTID = *value*
サービス・プロバイダーで、JSON メッセージを表すために使用される最上位のデータ構造を保持するコンテナーの名前を指定します。

CICS がターゲット・アプリケーション・プログラムに渡すコンテナーの長さは、要求コンテナーの長さおよび応答コンテナーの長さより大きくなります。

DATA-SCREENING = { **ENABLED** | **DISABLED** }
アプリケーション提供のデータのエラーを検査するかどうかを指定します。

ENABLED
言語構造と矛盾するアプリケーション提供のランタイム・データは、エラーとして扱われ、メッセージ DFHPI1010 が発行されます。エラー応答がアプリケーションに 返されます。

DISABLED
言語構造と矛盾するアプリケーション提供のランタイム・データの値は、 デフォルト値で置き換えられます。例えば、数値フィールド内のゼロは誤った値を置き換えます。メッセージ DFHPI1010 は発行されず、通常の応答がアプリケーションに返されます。この機能を使用して、 初期設定されていない出力フィールドから生成される INVALID_PACKED_DEC および INVALID_ZONED_DEC エラー応答を回避できます。

DATA-TRUNCATION = { **DISABLED** | **ENABLED** }
固定長フィールド構造で可変長データを許容するかどうかを指定します。

DISABLED
データが CICS が予期する固定長より短い場合に、CICS は切り捨てられたデータを拒否してエラー・メッセージを発行します。

ENABLED
データが CICS が予期する固定長より短い場合に、CICS は切り捨てられたデータを許容して、欠落データをヌル値として処理します。

DATETIME = **PACKED15** | **STRING**
JSON 日時エレメントを言語構造にマップする方法を指定します。

PACKED15
デフォルトでは、すべての JSON 日時エレメントがタイム・スタンプとして処理され、CICS ABSTIME 形式にマップされます。

STRING
JSON 日時エレメントはテキストとして処理されます。

DEFAULT-ARRAY-MAXITEMS = *value*
JSON スキーマで最大出現回数の情報 (maxItems) が示されていない場合に適用する最大配列境界を指定します。このパラメーターが設定されていない場合、最大制限は適用されません。このパラメーターの値は、1 から 2147483647 までの範囲の正整数です。このパラメーターを **INLINE-MAXOCCURS-LIMIT** パラメーターとともに使用することによって、 JSON 配列が言語構造にマップされる方法を制御することができます。

DEFAULT-CHAR-MAXLENGTH = **255** | *value*
マッピング・レベルが 1.2 以上のとき、Web サービス記述文書に長さが暗黙指定されていない場合に、マッピングについて文字データのデフォルトの配列長を文字数で指定します。 このパラメーターの value は、1 から 2,147,483,647 の範囲の正整数です。

DEFAULT-FRACTION-DIGITS = { **3** | *value* }
JSON 10 進スキーマ・タイプで使用するデフォルトの小数桁数を指定します。デフォルト値は 3 です。 COBOL の場合、有効な範囲は 0 から 17、またはパラメーター **WIDE-COMP3** が使用されている場合、0 から 30 です。 C または PLI の場合、有効な範囲は 0 から 30 までです。

HTTP-METHODS = **{** *GET* | *POST* | *PUT* | *DELETE* | *HEAD* **},{** *GET* | *POST* | *PUT* | *DELETE* | *HEAD* **},***
これはオプションのパラメーターです。

デフォルト値には、GET、POST、PUT、および DELETE が設定されます。これは、アプリケーションが 4 つの主な RESTful 操作をサポートすることを DFHJS2LS に示します。
値が指定されると、DFHJS2LS は、明示的に指定される HTTP メソッドのみを受け入れる WSBind ファイルを作成します。
アプリケーションが HEAD メソッドを実装する場合、そうすることを明示的に選択する必要があります。デフォルトでは、DFHJS2LS は、アプリケーションが着信 HTTP HEAD メソッドをサポートしないことを想定しています。
JSON クライアントがサポートされない HTTP メソッドを使用しようとすると、CICS は HTTP 405 応答で応答します。

INLINE-MAXOCCURS-LIMIT = **1** | *value*
インラインの可変反復内容を maxItems JSON スキーマ・キーワードに基づいて使用するかどうかを指定します。インラインにマップされる可変の繰り返しコンテンツは、生成される言語構造の残りの部分と共に現在のコンテナーに入ります。 可変の繰り返しコンテンツは、2 つの部分 (データの出現回数を格納するカウンターと、データのそれぞれの出現を格納する配列) に分けて格納されます。 可変繰り返しコンテンツに代わるマッピングとして、コンテナー・ベースのマッピングがあります。この場合、データの出現回数、およびデータを収容するコンテナーの名前を格納します。別個のコンテナーにデータを格納するとパフォーマンスに影響を与えるため、インライン・マッピングの方が適しているかもしれません。

INLINE-MAXOCCURS-LIMIT パラメーターは、マッピング・レベル 2.1 以降でのみ使用できます。 **INLINE-MAXOCCURS-LIMIT** の値は、 0 から 32,767 の範囲の正整数です。 値 0 は、インライン・マッピングを使用しないことを示します。

値 1 を使用すると、オプション・エレメントがインラインでマップされます。maxOccurs 属性の *value* が **INLINE-MAXOCCURS-LIMIT** の *value* より大きい場合、コンテナーに基づくマップングが使用されます。それ以外の場合はインライン・マッピングが使用されます。

可変の繰り返しリストをインラインでマップするかどうか決定する際には、繰り返されるデータの単一項目の長さを考慮してください。 長い項目が少ない回数だけ出現する場合は、コンテナーに基づくマッピングが適しています。短い項目が数多く出現する場合は、インライン・マッピングが適しています。

JSON-SCHEMA-RESTFUL = *value*
これは必須パラメーターです。

この値は、応答 JSON スキーマが格納される UNIX システム・サービスの場所を示します。

LANG = **COBOL**|**PLI-ENTERPRISE**|**PLI-OTHER**|**C**|**CPP**
高水準言語構造のプログラミング言語を指定します。

COBOL
COBOL

PLI-ENTERPRISE
Enterprise PL/I

PLI-OTHER
Enterprise PL/I 以外の PL/I のレベル

C C

CPP C++

LOGFILE = *value*
DFHJS2LS 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 z/OS UNIX 名です。 DFHJS2LS は、このファイルが存在しない場合、ディレクトリー構造は作成しませんがファイルを作成します。

通常はこのファイルを使用しませんが、DFHJS2LS 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。

MAPPING-LEVEL = **{ 1.0 | 1.1 | 1.2 | 2.0 | 2.1 | 2.2 | 3.0 | 4.0 | 4.1 | 4.2 | 4.3 }**
Web サービス・バインディング・ファイルおよび言語構造を生成する際に、DFHJS2LS が使用するマッピングのレベルを指定します。 以下のオプションを選択できます。

1.0 このオプションは、SOAP Web サービスとの互換性のために保持されています。 JSON で使用することは推奨されていません。

1.1 このオプションは、SOAP Web サービスとの互換性のために保持されています。 JSON で使用することは推奨されていません。

1.2 このオプションは、SOAP Web サービスとの互換性のために保持されています。 JSON で使用することは推奨されていません。

2.0 このオプションは、SOAP Web サービスとの互換性のために保持されています。 JSON で使用することは推奨されていません。

2.1 このオプションは、SOAP Web サービスとの互換性のために保持されています。 JSON で使用することは推奨されていません。

2.2 このオプションは、SOAP Web サービスとの互換性のために保持されています。 JSON で使用することは推奨されていません。

3.0 このマッピング・レベルは、使用可能なすべてのオプション・セットを使用して JSON スキーマを生成する場合に使用します。

4.0 UTF-16 を使用する場合には、CICS TS 5.2 以降の領域でこのマッピング・レベルを使用します。

4.1 切り捨て可能な配列をサポートするには、CICS TS 5.2 以降の領域でこのマッピング・レベルを使用します。

4.2 追加プロパティーについては、CICS TS 5.4 以降の領域でこのマッピング・レベルを使用します。

4.3 多次元配列をサポートするには、CICS TS 5.4 以降の領域でこのマッピング・レベルを使用します。

マッピング・レベルについて詳しくは、CICS JSON アシスタントのマッピング・レベルを参照してください。

MAPPING-OVERRIDES = { SAME-AS-MAPPING-LEVEL |HYPHENS-AS-UNDERSCORES | INTEGER-AS-PIC9 | LESS-DUP-NAMES | UNDERSCORES-AS-HYPHENS | NO-ARRAY-NAME-INDEXING }

言語構造を生成するときの指定されたマッピング・レベルについて、デフォルトの動作を指定変更するかどうかを指定します。

SAME-AS-MAPPING-LEVEL
このパラメーターは、マッピング・レベルと同じスタイルで言語構造を生成します。これはデフォルトです。

HYPHENS-AS-UNDERSCORES
PL/I のみ。このパラメーターは、生成される PL/I 言語構造を読みやすくするために、JSON スキーマ内のすべてのハイフンを文字 X ではなく下線に変換します。詳しくは、JSON スキーマから PL/I へのマッピングを参照してください。このオプションは、マッピング・レベル 4.2 で自動的に有効になります。

INTEGER-AS-PIC9
COBOL および DFHJS2LS のみ。このパラメーターは、JSON スキーマからの整数値を、英数字ではなく数字として含む言語構造を生成します。このオプションは、マッピング・レベル 4.0 で自動的に有効になります。

LESS-DUP-NAMES
このパラメーターは、フィールドを直接参照できるように名前の末尾に `_value` を付けた非構造性の構造フィールド名を生成します。例えば、以下の PL/I 言語構造で、MAPPING-OVERRIDES=LESS-DUP-NAMES が指定されるとき、レベル 12 フィールドの streetName には接尾部に `_value` が付いています。

```
09 streetName,
12 streetName CHAR(255) VARYING
UNALIGNED,
12 filler BIT (7),
12 attr_nil_streetName_value BIT (1),
```

結果構造は以下のとおりです。

```
09 streetName,
12 streetName_value CHAR(255) VARYING
UNALIGNED,
12 filler BIT (7),
12 attr_nil_streetName_value BIT (1),
```

このオプションは、マッピング・レベル 4.2 で自動的に有効になります。

UNDERSCORES-AS-HYPHENS

COBOL のみ。 このパラメーターは、生成される COBOL 言語構造を読みやすくするために、JSON スキーマ内のすべての下線を文字 X ではなくハイフンに変換します。フィールド名の競合が発生する場合、そのフィールドが必ず固有になるように番号が付きます。 詳しくは、JSON スキーマから COBOL へのマッピングを参照してください。

このオプションは、マッピング・レベル 4.0 で自動的に有効になります。

NO-ARRAY-NAME-INDEXING

COBOL および Enterprise PL/I のみ。配列内のフィールド名が上位構造のスコープ内でのみ固有であるようにします。

MINIMUM-RUNTIME-LEVEL = { MINIMUM | 1.0 | 1.1 | 1.2 | 2.0 | 2.1 | 2.2 | 3.0 | 4.0 | 4.1 | 4.2 | 4.3 | CURRENT }

Web サービス・バインディング・ファイルを配置できる最小の CICS 実行時環境を指定します。指定した他のパラメーターと一致しないレベルを選択すると、エラー・メッセージが送信されます。 以下のオプションを選択できます。

MINIMUM

選択したパラメーターを前提として、最低限可能な CICS 実行時レベルが自動的に割り振られます。

1.0 このオプションは、SOAP Web サービスとの互換性のために保持されています。 JSON で使用することは推奨されていません。

1.1 このオプションは、SOAP Web サービスとの互換性のために保持されています。 JSON で使用することは推奨されていません。

1.2 このオプションは、SOAP Web サービスとの互換性のために保持されています。 JSON で使用することは推奨されていません。

2.0 このオプションは、SOAP Web サービスとの互換性のために保持されています。 JSON で使用することは推奨されていません。

2.1 このオプションは、SOAP Web サービスとの互換性のために保持されています。 JSON で使用することは推奨されていません。

2.2 このオプションは、SOAP Web サービスとの互換性のために保持されています。 JSON で使用することは推奨されていません。

3.0 生成された Web サービス・バインディング・ファイルは、CICS TS 4.1 以降の領域にデプロイされます。

注: JSON サポートが使用可能なのは、CICS TS 4.2 以降に限られます。

4.0 生成された Web サービス・バインディング・ファイルは、CICS TS 5.2 以降の領域に正常にデプロイされます。この実行時レベルでは、**MAPPING-LEVEL** パラメーターにマッピング・レベル 4.0 以前を使用できます。このレベルでは任意のオプション・パラメーターを使用できます。

4.1 生成された Web サービス・バインディング・ファイルは、CICS TS 5.2 以降の領域に正常にデプロイされます。この実行時レベルでは、**MAPPING-LEVEL** パラメーターには、マッピング・レベル 4.1 以前を使用することができます。

4.2 生成された Web サービス・バインディング・ファイルは、CICS TS 5.4 以降の領域に正常にデプロイされます。この実行時レベルでは、**MAPPING-LEVEL** パラメーターには、マッピング・レベル 4.2 以前を使用することができます。

4.3 生成された Web サービス・バインディング・ファイルは、CICS TS 5.4 以降の領域に正常にデプロイされます。この実行時レベルでは、**MAPPING-LEVEL** パラメーターには、マッピング・レベル 4.3 以前を使用することができます。

CURRENT
生成された Web サービス・バインディング・ファイルは、Web サービス・バインディング・ファイルの生成に使用するものと同じ実行時レベルで、CICS 領域に正常に配置されます。

NAME-TRUNCATION = { LEFT | RIGHT }
JSON 名が左と右のどちらから切り捨てられるかを指定します。 CICS Web サービス・アシスタントは、指定された高水準言語における適切な長さに JSON 名を切り捨てます。デフォルトでは、名前は右から切り捨てられます。

PDSCP = *value*
PDSMEM パラメーターで指定されている区分データ・セット・メンバーで使用されるコード・ページを指定します (*value* は CCSID 番号または Java コード・ページ番号)。このパラメーターを指定しない場合は、z/OS UNIX システム・サービスのコード・ページが使用されます。 例えば、**PDSCP**=037 と指定できます。

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、**PDSMEM** パラメーターで指定されます。

PDSMEM = *value*
高水準言語構造が含まれる区分データ・セット・メンバーの名前を生成するために DFHJS2LS が使用する、1 から 6 文字の文字接頭部を指定します。

DFHJS2LS は接頭部に 01 を付加することにより、区分データ・セット・メンバーを生成します。

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。

ターゲット・アプリケーション・プログラムが要求を処理するとき、同じメカニズムを使用して応答を戻す必要があります。要求が通信域で受信されている場合は、応答を通信域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、応答をコンテナーに戻す必要があります。CICS がターゲット・アプリケーション・プログラムに渡す通信域またはコンテナーの長さは、要求の通信域またはコンテナーの長さ、および応答の通信域またはコンテナーの長さより大きくなります。

PGMNAME = *value*
CICS PROGRAM リソース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 DFHJS2LS 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・プログラムのリソース名を指定します。

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 DFHJS2LS を使用する場合、このパラメーターを省略します。

STRUCTURE = *name*
C と C++ の場合にのみ、構造体の名前を生成する方法を指定します。

生成される構造体には *name 01* という名前が付きます。
名前が省略される場合、構造体の名前は、指定した **PDSMEM** パラメーターから生成される区分データ・セット・メンバー名と同じになります。

SYNCONRETURN = { **NO** | **YES** }
リモート Web サービスが同期点を発行できるかどうかを指定します。

NO リモート Web サービスは同期点を発行できません。 この値はデフォルトです。リモート Web サービスが同期点を発行すると、ADPL 異常終了になり失敗します。

YES リモート Web サービスは同期点を発行できます。 YES を選択した場合、リモート Web サービスから制御が戻されたときにリモート・タスクが別個の作業単位としてコミットされます。リモート Web サービスがリカバリー可能リソースを更新して戻した後に障害が発生した場合、そのリソースの更新内容をバックアウトすることはできません。

TRANSACTION = *name*
サービス・プロバイダーで、このパラメーターは、応答を組み立てるためにパイプラインを開始できる 1 から 4 文字の別名トランザクションの名前を指定しま

す。 このパラメーターの値は、URIMAP リソースが **PIPELINE** スキャン・コマンドを使用して自動的に作成される際に、URIMAP リソースの TRANSACTION 属性を定義するために使用されます。

許容文字:
A-Z a-z 0-9 $ @ # _ < >

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相対 URI を指定します。 CICS は、DFHJS2LS 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 のパスのコンポーネントを指定します。 URI の終わりにワイルドカード * を使用するときには、URI 値の後にコンマを続ける必要があります。

USERID = *id*
サービス・プロバイダーでは、このパラメーターは、任意の Web クライアントで使用可能な 1 文字から 8 文字のユーザー ID を指定します。 アプリケーションから生成される応答や Web サービスについては、そのユーザー ID の下で別名トランザクションが追加されます。 このパラメーターの値は、URIMAP リソースが **PIPELINE** スキャン・コマンドを使用して自動的に作成される際に、URIMAP リソースの USERID 属性を定義するために使用されます。

許容文字:
A-Z a-z 0-9 $ @ #

WIDE-COMP3 = { **FULL** | **NO** | **YES** }
生成される COBOL または PL/I 言語構造でパック 10 進数の可変長の最大サイズを制御します。

FULL COBOL および PL/I の場合。DFHJS2LS は、すべての有効な値を保持できるサイズでパック 10 進数フィールドを生成します。最大サイズは 31 桁です。これはデフォルトです。

NO COBOL のみ。 DFHJS2LS は、COBOL 言語構造タイプ COMP-3 を生成しているとき、パック 10 進数の可変長を 18 に制限します。 パック 10 進数のサイズが 18 より大きい場合、指定されたタイプは合計 18 桁に制限されていることを示すメッセージ DFHPI9022W が出されます。

YES COBOL のみ。 DFHJS2LS は、COBOL 言語構造タイプ COMP-3 を生成しているとき、最大サイズ 31 をサポートします。

注**:** NO および YES オプションを指定すると、すべての有効な値を表わすことができないフィールドが生成されます。FULL オプションは、この問題を回避します。ただし、FULL オプションを指定すると、パック 10 進数フィールドで無効な値が表される可能性があります。例えば、スキーマに最大サイズとして最大 5 桁の数字と最大 2 桁の小数点以下の数値が指定されている場合、FULL オプションを指定すると、7 桁の数字が許容されるパック 10 進数フィールドが生成さ

れます。この場合、25000 や 999.99 などの有効な値を収容するだけのスペースがあると同時に、9999.99 などの無効な値を収容するだけのスペースも提供されることになります。FULL オプションを使用する場合は、アプリケーション・データに無効な値が生成されないよう注意してください。

WSBIND = *value*
Web サービス・バインディング・ファイルの完全修飾 z/OS UNIX 名です。DFHJS2LS は、このファイルが存在しない場合、ディレクトリー構造は作成しませんがファイルを作成します。デフォルトのファイル拡張子は .wsbind です。

その他の情報

- DFHJS2LS を実行するユーザー ID は、UNIX システム・サービスを使用するように構成されていなければなりません。 ユーザー ID には、CICS z/OS UNIX ファイル構造および PDS ライブラリーに対する読み取り権限、および **LOGFILE**、**WSBIND**、および **WSDL** パラメーターで指定されたディレクトリーへの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。
- JCL の最大パラメーター長は 100 文字です。 **STDPARM** ステートメントを使用してこの値を増やすことができます。詳細については、z/OS UNIX システム・サービス ユーザーズ・ガイドを参照してください。

例

```
//JS2LS JOB '
accounting information
',
name,MSGCLASS=A
// SET QT=''''
//JAVAPROG EXEC DFHJS2LS,
// TMPFILE=&QT.&SYSUID.&QT
/INPUT.SYSUT1 DD *
PDSLIB=//CICSHLQ.SDFHSAMP
PDSMEM=CPYBK2
JSON-SCHEMA-RESTFUL=example.json
LANG=COBOL
LOGFILE=/u/exampleapp/wsbind/example.log
MAPPING-LEVEL=4.0
CHAR-VARYING=NULL
INLINE-MAXOCCURS-LIMIT=2
PGMNAME=DFH0XCMN
URI=exampleApp/example/*,
PGMINT=COMMAREA
SYNCONRETURN=YES
WSBIND=/u/exampleapp/wsbind/example.wsbind
/*
```

JSON アシスタントを使用した JSON サービス・プロバイダー・アプリケーションの作成

JSON スキーマ v4 (ドラフト) に準拠する JSON スキーマから、または高水準言語データ構造から、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。CICS JSON 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デプロイを支援します。

このタスクについて

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。

- JSON スキーマから開始し、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 サービス記述に準拠するサービス・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、アシスタントを使用して JSON スキーマを生成する。

  既存のプログラムを JSON サービスとして公開する場合は、このオプションを使用します。

JSON スキーマからのサービス・プロバイダー・アプリケーションの作成

CICS JSON アシスタントを使用すると、JSON スキーマからサービス・プロバイダー・アプリケーションを作成できます。

始める前に

サービス・プロバイダー・アプリケーションを作成する前に、以下の条件が満たされていなければなりません。

- Web サービス記述が z/OS の UNIX ファイルに含まれている必要があり、適切なプロバイダー・モード・パイプラインを CICS 領域に作成する必要があります。
- DFHJS2LS 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、z/OS UNIX ライブラリーと PDS ライブラリーに対する読み取り権限と、**LOGFILE**、**WSBIND**、**JSON-SCHEMA-REQUEST**、**JSON-SCHEMA-RESPONSE**、および **JSON-SCHEMA-RESTFUL** 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。
- このユーザー ID で Java を実行するためには、この ID に十分なストレージを割り振る必要があります。 サポートされている任意のバージョンの Java を使用できます。 DFHJS2LS はデフォルトでは、**JAVADIR** パラメーターで指定されている Java バージョンを使用します。

このタスクについて

JSON アシスタントを使用すると、JSON スキーマからサービス・プロバイダー・アプリケーション用の言語構造を作成できます。

手順

1. Web サービス・バインディング・ファイルおよび 1 つ以上の言語データ構造を生成する場合は、DFHJS2LS バッチ・プログラムを使用します。 DFHJS2LS には、アプリケーションで必要なバインディング・ファイルと言語構造を作成する

際に柔軟性を提供する、多数のオプション・パラメーターが含まれています。既存のアプリケーションを Web サービスで使用可能にするときは、以下のオプションを検討してください。

- サービス・プロバイダー・アプリケーション・プログラムにデータを渡すために CICS が使用すべきメカニズムは? チャネルを使用してコンテナー内のデータを渡すか、COMMAREA を使用することができます。 チャネルとコンテナーが推奨されます。 これは、**PGMINT** パラメーターを使用して指定します。
- 生成する言語は? DFHJS2LS は、COBOL、C/C++、または PL/I 言語データ構造を生成できます。言語は、**LANG** パラメーターを使用して指定します。
- 使用するマッピング・レベルは? マッピング・レベルが高いほど、実行時に文字とバイナリー・データの処理の制御とサポートを行いやすくなります。一部のオプション・パラメーターは、高いマッピング・レベルでしか使用できません。使用できる最高のマッピング・レベルを使用することをお勧めします。マッピング・レベルは **MAPPING-LEVEL** パラメーターで指定します。
- Web サービス・リクエスターで使用する URI は? **URI** パラメーターを使用して相対 URI を指定します。例えば、`URI=/my/test/webservice` です。 この値は、URIMAP リソースの作成時に CICS によって使用されます。
- Web サービス要求と応答を実行するためのトランザクションとユーザー ID は? 別名トランザクションを使用すると、サービス・リクエスターへの応答を構成するためのアプリケーションを実行することができます。 別名トランザクションは、ユーザー ID を基にして接続されます。これは、**TRANSACTION** パラメーターと **USERID** パラメーターを使用して指定します。これらの値は、URIMAP リソースの作成時に使用されます。 特定のトランザクションを使用したくない場合は、これらのパラメーターを使用しないでください。

DFHJS2LS を実行依頼すると、CICS は Web サービス・バインディング・ファイルを生成して、ユーザーが **WSBIND** パラメーターを使用して指定した場所に置きます。言語構造は、ユーザーが **PDSLIB** パラメーターを使用して指定した区分データ・セットに置かれます。

2. 生成された 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プロバイダー・モード PIPELINE リソースの `pickup` ディレクトリーにコピーします。 バインディング・ファイルはバイナリー・モードでコピーする必要があります。
3. 生成された言語構造とインターフェースを取るサービス・プロバイダー・アプリケーション・プログラムを作成して、必要なビジネス・ロジックを実装します。
4. **PIPELINE SCAN** コマンドを使用して、WEBSERVICE リソースおよび URIMAP リソースを動的に作成します。
   - WEBSERVICE リソースは、CICS で Web サービス・バインディング・ファイルをカプセル化し、実行時に使用されます。
   - URIMAP リソースは、WEBSERVICE リソースを特定の URI に関連付けるための情報を CICS に提供します。

   別の方法として、リソースを自分で定義することもできますが、これは推奨されていません。

タスクの結果

DFHJS2LS の実行依頼時に問題が発生した場合や、リソースが正しくインストールされない場合は、JSON アシスタントのトラブルシューティングを参照してください。

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して、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します。

始める前に

サービス・プロバイダー・アプリケーションを作成する前に、セットアップで以下の前提条件が満たされていることを確認します。

- 高水準言語データ構造は、次の基準を満たす必要があります。
  - データ構造は、例えば COBOL コピーブック内など、ソース・プログラムとは別個に定義される必要があります。
  - PL/I または COBOL 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る必要があります。 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する必要があります。

    C および C++ では、区分データ・セット内の同じメンバーにデータ構造を置くことができます。
- 言語構造が区分データ・セットで使用可能になっている必要があり、適切な PIPELINE リソースを CICS 領域内で作成する必要があります。
- DFHLS2JS 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、z/OS UNIX ライブラリーと PDS ライブラリーに対する読み取り権限と、**LOGFILE**、**WSBIND**、**JSON-SCHEMA-REQUEST**、および **JSON-SCHEMA-RESPONSE** 出力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。サポートされている任意のバージョンの Java を使用できます。デフォルトで、DFHLS2JS は **JAVADIR** パラメーターに指定されている Java バージョンを使用します。

手順

高水準データ構造を基にしたサービス・プロバイダー・アプリケーションを作成するには、以下のステップを実行します。

1. サービス・プロバイダー・アプリケーション・インターフェースがチャネルおよび多数のコンテナーを使用する場合には、JSON でインターフェースを記述するチャネル記述文書を作成してください。 チャネル記述文書を z/OS UNIX 上の適切なディレクトリーに保管する必要があります。 CICS はこの文書を使用

して、チャネル上のコンテナーから JSON メッセージを構成および分解します。 あるいは、チャネル記述文書を作成せずに、チャネルにある 1 つのコンテナーを使用することもできます。

チャネル記述文書を作成する方法について、詳しくは、 317 ページの『チャネル記述文書の作成』を参照してください。

2. 言語構造を基にして Web サービス・バインディング・ファイルおよび Web サービス記述を生成する場合は、DFHLS2JS バッチ・プログラムを使用します。 DFHLS2JS バッチ・プログラムは、*HLQ* .XDFHINST にあります。ここで、*HLQ* は CICS をインストールした場所です。 DFHLS2JS には、アプリケーションで必要なバインディング・ファイルと言語構造を作成する際に柔軟性を提供する、多数のオプション・パラメーターが含まれています。 既存のアプリケーションで Web サービスを使用可能にする際に、次のオプションについて検討してください。
   - サービス・プロバイダー・アプリケーション・プログラムにデータを渡すために CICS で使用するメカニズムは? チャネルを使用してコンテナー内のデータを渡すか、COMMAREA を使用することができます。メカニズムは、**PGMINT** パラメーターを使用して指定します。アプリケーション・インターフェースがチャネルおよび多数のコンテナーを使用する場合には、**REQUEST-CHANNEL** パラメーターを指定し、オプションで **RESPONSE-CHANNEL** を指定します。これらのパラメーターは、マッピング・レベルが 3.0 以上の場合にのみ使用できます。
   - 使用するマッピング・レベルは? マッピング・レベルが高いほど、実行時に文字とバイナリー・データの処理の制御とサポートを行いやすくなります。一部のオプション・パラメーターは、高いマッピング・レベルでしか使用できません。**MAPPING-LEVEL** パラメーターで使用できる最も高いマッピング・レベルを指定する必要があります。
   - Web サービスで使用する URI は? **URI** パラメーターを使用して絶対 URI を指定します。例えば、**URI**=http://www.example.org:80/my/test/webservice です。このアドレスの相対部分 (/my/test/webservice) は、URIMAP リソースの作成時に使用されます。

   DFHLS2JS を実行依頼すると、CICS は Web サービス・バインディング・ファイルを生成して、ユーザーが **WSBIND** パラメーターを使用して指定した場所に置きます。 生成された JSON スキーマは、ユーザーが **JSON-SCHEMA-REQUEST** パラメーターおよび **JSON-SCHEMA-RESPONSE** パラメーターを使用して指定した場所に置かれます。
3. 生成された JSON スキーマを確認します。

   これらのスキーマを使用して、JSON Web サービスの入出力データ・フォーマットを定義します。アプリケーション開発者は、JSON Web サービスと相互作用するアプリケーションを作成する際にこれらのスキーマを使用する必要があります。

   注: 生成されたスキーマを変更すると、生成された Web サービス・バインディング・ファイル WSBind が無効になります。

   スキーマを変更する場合、例えばスキーマ内のフィールドを名前変更する場合などには、DFHJS2LS を使用して新しい Web サービス・バインディング・ファ

イルと、一連の新しい言語構造を生成しなければなりません。新しい言語構造を使用するためには、CICS 内のアプリケーション・プログラムを変更する必要があります。

4. 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プロバイダー・モード・パイプラインのピックアップ・ディレクトリーにコピーします。 Web サービス・バインディング・ファイルはバイナリー・モードでコピーする必要があります。
5. **PIPELINE SCAN** コマンドを使用して、WEBSERVICE リソースおよび URIMAP リソースを動的に作成します。
   - WEBSERVICE リソースには CICS で Web サービス・バインディング・ファイルが含まれており、実行時に使用されます。
   - URIMAP リソースは、WEBSERVICE リソースを特定の URI に関連付けるための情報を CICS に提供します。

   別の方法として、リソースを自分で定義することもできます。

タスクの結果

サービス・プロバイダー・アプリケーションの作成は完了です。

DFHLS2JS の実行依頼時に問題が発生した場合や、リソースが正しくインストールされない場合は、JSON アシスタントのトラブルシューティングを参照してください。

次のタスク

サービスにアクセスする Web サービスを作成するすべてのユーザーが、この Web サービス記述を使用できるようにします。

チャネル記述文書の作成

サービス・プロバイダー・アプリケーションで多数のコンテナーを持つチャネル・インターフェースを使用する場合、チャネル記述文書を作成します。

このタスクについて

XML エディターを使用して、チャネル記述文書を作成します。チャネル記述のスキーマは `channel.xsd` という名前で、`/usr/lpp/cicsts/cicsts55/schemas/channel` ディレクトリー (`/usr/lpp/cicsts/cicsts55` は z/OS UNIX 上の CICS ファイルのデフォルト・インストール・ディレクトリー) にあります。

手順

1. 次のようにして、<channel> エレメントおよび CICS チャネル名前空間を持つ XML 文書を作成します。

```
<channel name="myChannel"
         xmlns="http://www.ibm.com/xmlns/prod/CICS/channel">
</channel>
```

2. アプリケーション・プログラム・インターフェースがチャネルで使用するコンテナーごとに 1 つの <container> エレメントを追加します。 それぞれのコンテナーを記述する名前、タイプ、および使用属性を使用する必要があります。 以下の例は、異なる属性値を持つ 6 つのコンテナーを示しています。

```
<container name="cont1" type="char" use="required"/>
<container name="cont2" type="char" use="optional"/>
<container name="cont3" type="bit" use="required"/>
<container name="cont4" type="bit" use="optional"/>
<container name="cont5" type="bit" use="required">
  <structure location="//HLQ.PDSNAME(MEMBER)"/>
</container>
<container name="cont6" type="bit" use="optional">
  <structure location="//HLQ.PDSNAME(MEMBER2)"/>
</container>
```

この structure エレメントは、区分データ・セット・メンバーにある言語構造で内容が定義されていることを示します。

3. XML 文書を z/OS UNIX で保管します。

チャネル・スキーマ

チャネル記述文書は、以下のスキーマに従う必要があります。

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://www.ibm.com/xmlns/prod/CICS/channel"
        xmlns:tns="http://www.ibm.com/xmlns/prod/CICS/channel"
        elementFormDefault="qualified">
  <element name="channel">
    1
    <complexType>
      <sequence>
        <element name="container" maxOccurs="unbounded" "unbounded" minOccurs="0">
          2
          <complexType>
            <sequence>
              <element name="structure" minOccurs="0">
                3
                <complexType>
                  <attribute name="location" type="string" use="required"/>
                  <attribute name="structure" type="string" use="optional"/>
                </complexType>
              </element>
            </sequence>
            <attribute name="name" type="tns:name16Type" use="required"/>
            <attribute name="type" type="tns:typeType" use="required"/>
            <attribute name="use" type="tns:useType" use="required"/>
          </complexType>
        </element>
      </sequence>
      <attribute name="name" type="tns:name16Type" use="optional" />
    </complexType>
  </element>
  <simpleType name="name16Type">
    <restriction base="string">
      <maxLength value="16"/>
    </restriction>
  </simpleType>
  <simpleType name="typeType">
    <restriction base="string">
      <enumeration value="char"/>
      <enumeration value="bit"/>
    </restriction>
  </simpleType>
  <simpleType name="useType">
    <restriction base="string">
      <enumeration value="required"/>
      <enumeration value="optional"/>
    </restriction>
  </simpleType>
</schema>
```

1. このエレメントは CICS チャネルを表します。

2. このエレメントはチャネル内の 1 つの CICS コンテナーを表します。
3. structure は「ビット」モードのコンテナーでのみ使用できます。 location 属性は、コンテナーの内容をマップするファイルの場所を示します。 structure 属性を C および C++ で使用して、構造体の名前を示すことができます。

次のタスク

DFHLS2JS を実行して、Web サービス・プロバイダー・アプリケーション用のマッピングと JSON スキーマを作成します。 DFHLS2JS は、コンテナーがチャネル記述文書で指定されている順序で、チャネルのマッピングを JSON スキーマに入れます。

生成された JSON スキーマのカスタマイズ

DFHLS2JS によって生成される JSON スキーマには、自動的に生成される内容が含まれていますが、これらは公開前にユーザーが変更した方がよい場合があります。JSON スキーマをカスタマイズすると、Web サービス・バインディング・ファイルが再生成されることがあります。また、場合によっては、ラッパー・プログラムが作成されることもあります。

このタスクについて

以下の手順に従って、生成された JSON スキーマをカスタマイズします。

手順

1. HTTPS のサポートを公示する場合、DFHLS2JS で **URI** パラメーターを使用して、絶対 URI を設定します。
2. Web サービスのネットワークの場所を指定するには、DFHLS2JS で **URI** パラメーターを使用して、絶対 URI を設定します。
3. JSON スキーマの自動的に生成された名前が目的に合っているかどうかを検討します。 プロパティーの名前を変更できます。

   これらの値は、クライアント・プログラムをコーディングするプログラマチック・インターフェースの一部を形成します。 生成された名前に十分な意味がない場合、長期間にわたるアプリケーション・コードの保守が困難になる可能性があります。

   これらの値のいずれかを変更する場合は、DFHJS2LS を使用して Web サービス・バインディング・ファイルを再生成する必要があります。 生成される言語構造には、既存のアプリケーションとの互換性がないことが多く、アプリケーションを変更する必要がある場合があります。生成された言語構造を確認し、4の手順に説明されているように、ラッパー・プログラムを作成することを検討してください。
4. JSON スキーマで公開されている COMMAREA フィールドが適切かどうかを検討します。 次のように、JSON クライアント開発者にとって有益ではないフィールドを除去することができます。
   - 出力値のためにのみ使用されるフィールドを、入力データ構造をマップするスキーマから除去することができます。
   - 充てん文字フィールド。

- 自動的に生成される注釈

これらの変更を行う場合は、DFHJS2LS を使用して Web サービス・バインディング・ファイルを再生成する必要があります。 生成される新規の言語構造には、元の言語構造との互換性がないため、データを新規の表現から古い表現にマップするためのラッパー・プログラムを作成する必要があります。 このラッパー・プログラムは、ターゲット・アプリケーション・プログラムに対して **EXEC CICS LINK** コマンドを実行してから、戻されたデータをマップする必要があります。

このレベルのカスタマイズは最も多くの労力を必要としますが、JSON クライアント開発者にとって最も価値のあるプログラマチック・インターフェースにすることができます。

タスクの結果

ビジネス要件に一致するカスタマイズした JSON スキーマ、およびそれらを実装する CICS の PROGRAM が作成されました。

RESTful Web サービス・プロバイダー・アプリケーションの作成

RESTful JSON Web サービスの CICS 実装は、SOAP Web サービスの場合と似ています。ほとんどの概念とアーキテクチャーは共通ですが、CICS では JSON スキーマの使用が必要になります。

このタスクについて

RESTful JSON の Web サービスを実装するには、以下のタスクを行います。

手順

1. アプリケーション・インターフェースを生成します。

   入力:

   - RESTful Web サービス用のデータ・モデルを定義した JSON スキーマ。

   出力:

   - JSON スキーマをマップする言語構造 (COBOL コピーブックなど)。
   - WSBind ファイル。

   DFHJS2LS ユーティリティーを実行し、適切な入力パラメーターを指定します。以下のパラメーターがあります。

   - JSON スキーマの場所。
   - サポート対象メソッドのリスト (デフォルトでは、GET、PUT、POST および DELETE が使用可能です)。
   - サービスを配置する URI。
   - サービスを実装するアプリケーション PROGRAM の名前。
   - 必須のデータ・マッピング・パラメーター。
2. このインターフェースを使用するアプリケーションを作成します。

   入力:

   - ステップ 1 の言語構造

- 実装する RESTful 操作の認識

出力:

- CICS への配置に適したプログラム

以下を実行するプログラムを作成します。

注: DFHJS2LS に対して **CONTID** パラメーターで独自のコンテナーの名前を指定する場合には、以下ステップで示される DFHWS-DATA の代わりにこのコンテナーを使用する必要があります。

a. URI を検証し、リソースの実体を把握します。CICS は、対象となる URI のコンポーネントを特定するのに役立ついくつかのコンテナーを提供します。コンテナーについては、表 15で説明されています。例は、URI の各コンテナーの内容を示しています。

```
http://www.example.org:10000/JSONServices/CustomerDetails/13388?action=query
```

```
/JSONServices/CustomerDetails/* というパスに一致する URIMAP の場合
```

表 15. DFHWS-URI コンテナー: DFHWS-URI コンテナー

コンテナー	内容	例
DFHWS-URI	完全な URI	`http://www.example.org:10000/JSONServices/CustomerDetails/13388?action=query`
DFHWS-URIMAPPATH	URIMAP に一致する URI パスの部分	`/JSONServices/CustomerDetails/*`
DFHWS-URI-RESID	URIMAP で一致する部分を除いた URI パス (リソース *ID*)	`13388`
DFHWS-URI-QUERY	照会ストリング	`action=query`

b. URI を検証します。問題がある場合、その問題を CICS に報告し、終了します。

c. DFHHTTPMETHOD コンテナーに対する照会を実行し、駆動されているメソッドを判別します。詳しくは、DFHHTTPMETHODを参照してください。

d. POST (作成) メソッド (必要な場合):

- DFHWS-DATA コンテナーから入力データを読み取ります。
- ステップ 1 (320 ページ) で生成された言語構造を使用してデータを解釈します。
- データを検証します。問題がある場合、その問題を CICS に報告し、終了します。
- リソース を作成するために必要なアプリケーション固有プロセスを実行します。
- オプションで、クライアントに新しいリソースの ID を通知するために、DFHRESPONSE コンテナーに書き込みます。このコンテナーの内容

は、CICS では変換されませんが、HTTP 応答で直接送信されます。DFHWS-DATA コンテナーは無視されます。

e. GET (照会) または HEAD (照会) メソッドが必要な場合、リソース を表すデータを DFHWS-DATA コンテナーに書き込みます。

f. PUT (セット) メソッドが必要な場合:

- DFHWS-DATA コンテナーから入力データを読み取ります。
- ステップ 1 (320 ページ) で生成された言語構造を使用してデータを解釈します。
- データを検証します。問題がある場合、その問題を CICS に報告し、終了します。
- リソース を更新するために必要なアプリケーション固有プロセスを実行します。

g. DELETE メソッドが必要な場合、リソース を削除するために必要なアプリケーション固有プロセスを実行します。

注: CICS で実装された RESTful データ・モデルは、デフォルトでは PUT, POST, または DELETE メソッドのための応答本体送信は行いません。通常、RESTful アプリケーションは、成功か失敗かを示すために HTTP 状況コードを使用します。アプリケーションが正常に終了すると、CICS は、HTTP 応答 200 (OK) を送信します。送信エラー応答についての詳細は、アプリケーション・エラーの報告 を参照してください。 PUT, POST または DELETE メソッドのための応答本体を送信する場合は、DFHRESPONSE コンテナーを書き込む必要があります。それがある場合、CICS は追加プロセスなしで HTTP 本体にあるこのコンテナーの内容を送信します。CICS は、これらのメソッドのための応答プロセスの中で、DFHWS-DATA コンテナーを無視します。

3. 成果物を配置します。

入力:

- ステップ 1 (320 ページ) の WSBind ファイル。
- ステップ 2 (320 ページ) の CICS プログラム
- JSON 用に構成された CICS プロバイダー・モードのパイプライン・リソース。

出力:

- 配置された RESTful JSON Web サービス。

通常の方法でプログラムを CICS に配置します。

以下のいずれかの方法で行います。

- WSBind ファイルをパイプラインの「ピックアップ」ディレクトリーに配置してから、**PIPELINE SCAN** コマンドを発行して、WEBSERVICE リソースおよび URIMAP リソースを作成します。
- WEBSERVICE リソースと関連する URIMAP リソースを手動で定義およびインストールします。URIMAP は、URI を PIPELINE と WEBSERVICE の両方に関連付ける必要があります。

4. サービスをテストします。

入力:

- JSON スキーマ。
- RESTful web サービスの URI。
- ステップ 3 で配置したサービス。
- 任意の JSON テスト・クライアント。

出力:

- 正常に処理された要求。

任意のテスト・クライアントを使用して、JSON 要求を CICS に送信します。

予期しない応答を受け取る場合、問題の判別を試みてください。詳しくは、JSON 要求での問題のトラブルシューティングを参照してください。

RESTful Web サービス・プロバイダー・アプリケーションの設計上の考慮事項

このトピックでは、JSON に対して RESTful Web サービス・プロバイダー・アプリケーションを計画および設計する際に考慮する必要のあるいくつかの問題について説明します。

リソース・コレクション

RESTful API の一般的な設計では、リソース・コレクションの取得をサポートする必要があります。例えば、次のようにオブジェクトのセットを返すサービスが存在することがあります。

```
GET /Services/CustomerDetails?Surname=Cooper
```

この要求では、姓が "Cooper" であるすべての CustomerDetails オブジェクトに関する情報を返すことが期待されます。個々の CustomerDetails オブジェクトは、次のようなより詳細な URI を使用して返すことができます。

```
GET /Services/CustomerDetails/Customer27
```

この例では、Customer27 は特定のカスタマーの 1 次キーです。この 2 番目の照会からの出力は、CustomerDetails オブジェクトのインスタンスになります。最初の照会からの出力は、明確ではありません。CustomerDetails オブジェクトのリストを返すか、または CustomerDetails オブジェクトの URI のリストを戻します (クライアントはこれらを個別に取得していくことができます)。両方の規則は共通です。

CICS でコレクションを実装するために、データ・インスタンスのリストまたは URI のリストのいずれかを記述する JSON スキーマを作成します。サービスを作成し、通常どおりそのサービスを実装できます。この例では、GET メソッドのみを選択して、実装します。以下の例のように、クライアントが大きなデータ・セットで前方および後方にページングできるようにページ編集サービスを実装することを検討できます。

```
GET
/Services/CustomerDetails?startRecord=200&endRecord=225
```

CICS で 2 つの URIMAP リソース (および 2 つの WEBSERVICE リソース) を必要とする場合があります。 1 つはコレクションのルート URI 構造をマップし、もう 1 つはインスタンスのワイルドカード URIMAP になります。以下に例を示します。

```
URIMAP1: Path=/Services/CustomerDetails
WEBSERVICE=CollectionService
URIMAP2: Path=/Services/CustomerDetails/* WEBSERVICE=InstanceService
```

キャッシュ管理

RESTful アーキテクチャーは、標準 HTTP キャッシュ管理の手法の統合を促進します。このアーキテクチャーにより、GET 要求の結果がネットワークにキャッシュされ、サーバーの負荷が軽減されます。これを実現するためのメカニズムには、GET 要求に対して返されたデータの有効期限日時の設定が含まれます。

CICS におけるアプリケーション制御のキャッシュ有効期限をサポートする汎用目的のメカニズムはありませんが、EXEC CICS WEB WRITE HTTPHEADER API を使用して適切な HTTP ヘッダーを追加するために、パイプライン・ハンドラー・プログラムを作成することができます。アプリケーション・プログラムは、HTTP 要求を受け取るのと同じ CICS 領域でホストされている場合にのみ、類似した動作を実行できます。

DFHJS2LS 内のエレメントの可変配列

JSON には、エレメント数が変化する配列を含めることができます。一般に、エレメント数が変化する JSON スキーマは 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。 CICS はコンテナー・ベースのマッピングまたはインライン・マッピングを使用して JSON データにおけるエレメント数の変化に対応します。

エレメント数が変化する配列を JSON スキーマ内で表現するには、`"type"` 値が `"array"` のスキーマで、キーワード `minItems` と `maxItems` を使用します。

- `minItems` キーワードは、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。デフォルトは、値 0 です。
- `maxItems` キーワードは、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、`minItems` キーワードの値以上の任意の正の整数値を指定できます。
- `maxItems` キーワードがない場合、配列が無制限であることを示します。

可変配列 `"maxItems":1` によってオプション・フィールドを表すことができます。例えば、`"component"` という名前の 8 バイトのオプション・ストリングは次のようになります。

```
"properties":{
"component": {
"type":"array",
"maxItems":1,
"items": {
"type": "string",
"maxLength": 8
```

```
}
}
},
"required": ["component"]
```

次のように、"required" キーワード値にフィールド名を含めないことにより、同じ効果を得ることができます。

```
"properties":{
"component": {
"type": "string",
"maxLength": 8
}
}
```

一般に、エレメント数が変化する JSON スキーマは 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。 このような場合に対処するため、CICS は一連のコンテナーとしてアプリケーション・プログラム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。 これらの構造はアプリケーションへの入出力として使用されます。

- CICS が JSON データをアプリケーション・データに変換する場合、これらの構造にアプリケーション・データが追加され、アプリケーションがそのデータを読み取ります。
- CICS がアプリケーション・データを JSON データに変換する場合、アプリケーションによって構造内に追加されたアプリケーション・データが読み取られます。

以下の例は、こうしたデータ構造のフォーマットを示しています。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。

例 1. 固定のエレメント数

この例では、ちょうど 3 回発生するエレメントを示しています。

```
"properties":{
"component": {
"type": "array",
"maxItems": 3,
"minItems": 3,
"items": {
"type": "string",
"maxLength": 8
}
}
},
"required": ["component"]
```

この例では、エレメントが出現回数が事前に分かっているので、単純な COBOL 宣言、または他の言語のこれに相当するもので固定長配列として表すことができます。

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

例 2. マッピング・レベル 2 以下における変化するエレメント数

この例では、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。

```
"properties":{
"component": {
"type": "array",
"maxItems": 5,
"minItems": 1,
"items": {
"type": "string",
"maxLength": 8
}
}
},
"required": ["component"]
```

主データ構造には、2 つのフィールドの宣言が格納されます。 CICS が JSON データをバイナリー・データに変換すると、最初のフィールドである component-num にはエレメントがJSON データ内に現れる回数が格納され、2 番目のフィールドである component-cont には、コンテナーの名前が格納されます。

```
05 component-num PIC S9(9) COMP-5
05 component-cont PIC X(16)
```

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 DFHJS-component
02 component PIC X(8)
```

エレメントの出現回数を確認するには、component-num (1 から 5 の範囲の値が含まれる) の値を調べる必要があります。エレメント・コンテンツは component-cont で指定されたコンテナー内にあります。このコンテナーはエレメントの配列を保持しており、各エレメントは DFHJS-component データ構造によってマップされています。

minItems="0" (または未指定) で、かつ maxItems="1" の場合、このエレメントはオプションです。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、以下のように component-num 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 エレメントはなく、component-cont の内容は未定義です。
- この値が 1 の場合、component エレメントは component-cont 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、DFHJS-component データ構造によってマップされます。

注: JSON データが単一の繰り返しエレメントで構成される場合、DFHJS2LS は 2 つの言語構造を生成します。 主要な言語構造は、配列エレメントの数と、エレメントの配列を保持するコンテナーの名前を含んでいます。 2 番目の言語構造は、繰り返しエレメントの単一インスタンスをマップします。

例 3 マッピング・レベル 2.1 以上における変化するエレメント数

マッピング・レベル 2.1 では、CICS 支援機能で **INLINE-MAXOCCURS-LIMIT** パラメーターを使用できます。 **INLINE-MAXOCCURS-LIMIT** パラメーターは変化するエレメント数を処理する方法を指定します。 変化するエレメント数のマッピング・オプションは、 325 ページの『例 2. マッピング・レベル 2 以下における変化するエレメント数』に記述されているコンテナー・ベースのマッピング、またはインライン・マッピングです。 このパラメーターの値は、0 - 32767 までの範囲の正整数です。

- **INLINE-MAXOCCURS-LIMIT** のデフォルト値は 1 であり、オプション・エレメントがインラインで確実にマップされます。
- **INLINE-MAXOCCURS-LIMIT** パラメーターに値 0 が設定されている場合、インライン・マッピングは使用されません。
- maxItems が **INLINE-MAXOCCURS-LIMIT** の値以下の場合、インライン・マッピングが使用されます。
- maxItems が **INLINE-MAXOCCURS-LIMIT** の値以上の場合、コンテナー・ベースのマッピングが使用されます。

変化するエレメント数をインラインでマップすると、配列 (上記の例では発生するオカレンスが固定) およびカウンターが生成されます。 component-num フィールドは存在するエレメントのインスタンス数を示し、それらのインスタンスは配列によって示されます。 325 ページの『例 2. マッピング・レベル 2 以下における変化するエレメント数』で示される例では、**INLINE-MAXOCCURS-LIMIT** が 5 以下の場合に生成されるデータ構造は次のようになります。

```
05 component-num PIC S9(9) COMP-5 SYNC.
05 component OCCURS 5 PIC X(8).
```

最初のフィールド (component-num) は前のセクションで示されたコンテナー・ベース・マッピングの例の出力と同じです。 2 番目のフィールドには長さが 5 の配列があり、生成される可能性のある最大エレメント数を収容できる大きさです。

インライン・マッピングは、エレメントの出現回数およびデータが収容されるコンテナーの名前を格納するコンテナー・ベースのマッピングとは異なり、現行のコンテナーにあるすべてのデータを格納します。 現行のコンテナーにデータを格納するとパフォーマンスが向上するので、インライン・マッピングの方が一般的には望ましいといえます。

例 4. ネストされた変数配列

複雑な JSON スキーマには可変の繰り返しエレメントを含めることができ、そのエレメントにはさらに可変の繰り返しエレメントを含めることが可能です。この場合、記述される構造は、例で説明した 2 つのレベルを超えます。

この例は、1 回から 5 回出現する可能性がある必須エレメント "component1" 内でネストされたオプションのエレメント "component2" を示しています。

```
"properties":{
"component1": {
"type": "array",
"maxItems": 5,
"minItems": 1,
"items": {
"type": "object",
"properties":{
"component2":{
"type": "string",
"maxLength": 8
}
},
"required": ["component2"]
}
},
"required": ["component1"]
```

最上位のデータ構造は、前の例のデータ構造とまったく同じです。

```
05 component1-num PIC S9(9) COMP-5
05 component1-cont PIC X(16)
```

ただし、2 番目のデータ構造には、以下のエレメントが格納されます。

```
01 DFHJS-component1
02 component2-num PIC S9(9) COMP-5
02 component2-cont PIC X(16)
```

3 番目の構造には以下のエレメントが格納されます。

```
01 DFHJS-component2
02 component2 PIC X(8)
```

最外部エレメント "component1" の出現回数は、component1-num に格納されます。

component1-cont で指定されたコンテナーには、2 番目のデータ構造 (DFHJS-component1) のその数のインスタンスを持つ配列が含まれています。

component2-cont の各インスタンスは、異なるコンテナーを指定します。それぞれ、第 3 レベルの構造 (DFHJS-component2) によってマップされたデータ構造が含まれています。

この構造を説明するために、例と一致する次のような JSON データのフラグメントについて考えてみます。

```
{"component1":
[
{
"component2": "string1"
},
{
"component2": "string2"
},
]
}
```

"component1" は 3 回発生します。 最初の 2 つには、"component2" のインスタンスが含まれていますが、3 番目のインスタンスには含まれていません。

最上位のデータ構造では、component1-num に値 3 が含まれています。 component1-cont で指定されたコンテナーには、DFHJS-component1 の次の 3 つのインスタンスがあります。

1. 第 1 のインスタンスでは、component2-num の値は 1 で、component2-cont で指定されたコンテナーは *string1* を保持します。
2. 第 2 のインスタンスでは、component2-num の値は 1 で、component2-cont で指定されたコンテナーは *string2* を保持します。
3. 第 3 のインスタンスでは、component2-num の値は 0 で、component2-cont の内容は未定義です。

このインスタンスでは、完全なデータ構造は、次の合計 4 つのコンテナーによって表現されます。

- コンテナー DFHJS-DATA 内のルート・データ構造。
- component1-cont で指定されたコンテナー。

- component2-cont の最初の 2 つのインスタンスで指定された 2 つのコンテナー。

オプションの構造と **required** キーワード

データ構造は、JSON スキーマ "type" の "object" で定義します。"properties" キーワードで指定したオブジェクトを使用して、スキーマによりフィールド名が個々のタイプと関連付けられます。これらのフィールドが、JSON スキーマによって記述された JSON データの一部となることが必要かどうかは、"required" キーワードで指定される配列により制御されます。この配列には、JSON データ内に存在する必要があるすべてのフィールド名がリストされます。したがって、オプションのフィールドを表すには、そのフィールドをこの配列から除きます。あるいは、配列を空にはできないので、"required" キーワードを除きます。後者の場合、すべてのフィールドがオプションとなります。

オプションのフィールドは、0 個または 1 個のエレメントから成る変数配列として扱われます。これは、エレメント名に接尾部 "-num" が付加されたフィールドを追加します。全長が 28 文字を超える場合、エレメント名が切り捨てられます。これは実行時に、JSON データに値が存在していたことを示す場合は非ゼロ、そうでない場合はゼロになります。

以下の例に 2 つのフィールドを示します。一方は "required-structure" という必須フィールドで、もう一方は "optional-structure" というオプション・フィールドです。

```
{
"type": "object",
"properties": {
"required-structure": {
"type":"string",
"maxLength": 8
},
"optional-structure": {
"type":"string",
"maxLength": 8
}
},
"required": [
"required-structure"
]
}
```

生成される COBOL 構造には両方のフィールドが示されますが、2 番目の前にある "optional-structure-num" はエレメントに関する整数カウントで、0 は「なし」、1 は存在することを表しています。この値は、"optional-structure" に有効なデータが含まれるかどうかを示すように設定されます。

```
03 OutputData.
06 required-structure PIC X(8).
06 optional-structure-num PIC S9(9) COMP-5 SYNC.
06 optional-structure PIC X(8).
```

アプリケーション・エラーの報告

このトピックを読むと、RESTful JSON Web サービス・プロバイダー・アプリケーションがクライアントにエラーを報告する方法を理解できます。

トップダウン・シナリオでは、アプリケーションは多くの場合、エラー状態を報告する必要があります。例えば、「**アカウント番号を認識できません (Account Number not recognized)**」というエラーが発生する場合があります。この要件は、トップダウン・シナリオに固有のものです。ボトムアップ開発では、データ・フィールドにエンコードされるエラー・レポート・メカニズムをアプリケーションに組み込むか、アプリケーションが異常終了するかのいずれかです。トップダウン・シナリオでは、JSON スキーマがエラーを報告するフィールドを定義することはほとんどないため、代わりの方法が必要になります。

SOAP ベースの Web サービスの場合、この問題には **EXEC CICS SOAPFAULT** API を使用して対処します。 JSON には、SOAP 障害メッセージは存在しません。代わりに DFHHTTPSTATUS コンテナーを使用すると、JSON アプリケーションのアプリケーション検出エラーが報告されます。詳しくは、DFHHTTPSTATUSを参照してください。

注**:** さらに、必要に応じて DFHRESPONSE コンテナーや他の制御コンテナーをアプリケーションで使用して、より詳細なエラー応答を提供することもできます。

JSON Web サービスの制約事項

この参照資料では、JSON Web サービスでサポートされていない機能について説明します。

以下の機能はサポートされません。

- JSON を使用したリクエスター・モード・パイプラインはサポートされません。
- スキーマに照らした JSON データの実行時検証はサポートされていません。 JSON ペイロードで使用される WEBSERVICE リソースの VALIDATION 属性値は無視されます。
- JSON データにおける名前空間の使用 (Badgerfish または Mapped 規則) はサポートされていません。
- CICS に送られる JSON ペイロードは UTF-8 でエンコードされる必要があります。これ以外のエンコード方式はサポートされていません。同様に、CICS によって送信される JSON は常に UTF-8 でエンコードされます。
- JSON パイプラインを使用した WebSphere MQ トランスポートはサポートされていません。
- ベンダー変換プログラムを JSON 変換プログラムと共に使用することは、サポートされていません。
- JSON パイプラインで SOAP Web サービス・アプリケーション用に作成された WSBind ファイルを再使用することは、サポートされていません。 JSON サービス・プロバイダー・アプリケーションと共に使用される WSBind ファイルは、JSON アシスタントによって生成される必要があります。
- JSON ペイロードが CICS によって変換されるときに、必要な内容が一部欠落している場合、データ構造内のこれに相当するフィールドは、アプリケーション・プログラムに渡されるときに初期化されません。
- CICS では、signed long の最大値 (2^63 - 1) より大きい整数値を変換できません。ただし、整数値が引用符で囲まれている場合は変換できます。

- 単純なデータ・タイプの使用は、JSON スキーマのルートではサポートされていません。JSON スキーマは JSON オブジェクトまたは JSON 配列を記述します。とはいえ、 JSON オブジェクトには、単純なデータ・タイプ、配列、および他のオブジェクトを含めることができます。
- JSON スキーマで、**maxItems** の値を 1 として配列が宣言されている場合、CICS は 実行時に JSON を生成する際に、配列を単純なストリングまたは整数として直列化します。

重要: JSON プロパティー名では、先頭文字には `A-Z a-z _ :` のみがサポートされ、それ以降のすべての文字では `A-Z a-z 0-9 _ : . -` のみがサポートされます。

現在、Axis2 Web サービス・サポートには、アプリケーションとカスタマイズの開発および配置用の多数のオプションがあります。以下のオプションについてはサポートされていません。

- ユーザー提供のアプリケーション・ハンドラー - CICS 提供のアプリケーション・ハンドラー・クラス `com.ibm.cicsts.axis2.CICSAxis2ApplicationHandler` を使用する必要があります。
- ユーザー作成の Axis2 Java アプリケーション。
- SOAPFAULT API および WS-Addressing API は、JSON パイプラインでは使用できません。

コンテナーの制約事項

注: 一部のパイプライン・コンテナーには、JSON 要求の処理時にデータが取り込まれません。詳しくは、パイプラインで使用されるコンテナーを参照してください。

RESTful Web サービスにおける相違点

INQUIRE PIPELINE の場合:

- SOAPLEVEL は NOTSOAP を返します。
- MTOMNOXOPST、MTOMST、SENDMTOMST、SOAPRNUM、SOAPVNUM、XOPDIRECTST、XOPSUPPORTST の各属性は使用されません。

INQUIRE WEBSERVICE の場合:

- ARCHIVEFILE、BINDING、VALIDATIONST、XOPDIRECTST、XOPSUPPORTST の各属性は使用されません。
- WSDLFILE は、WEBSERVICE と関連付けられている JSON スキーマ・ファイルの名前を返します。

WEBSERVICE リソースの場合:

- ARCHIVEFILE パラメーターと VALIDATION パラメーターは使用されず、それらの値は無視されます。
- WSDLFILE は、WEBSERVICE と関連付けられている JSON スキーマ・ファイルの名前です。

アプリケーション・データおよび XML のマッピングと変換

アプリケーションのバイナリー・データを XML に、またはその逆の変換を行うアプリケーション・プログラムを作成できます。CICS では多数の高水準言語がサポートされており、ランタイム処理時のデータ変換方法をマップする XML アシスタントが提供されています。 CICS は Web サービス・サポートの一部として、同じテクノロジーを使用してアプリケーション・データを SOAP メッセージ内の XML にマップします。

始める前に

CICS によるデータまたは XML の変換時に、XML 支援機能や任意の妥当性検査を実行するには、Java をインストールしておく必要があります。

このタスクについて

アプリケーション・データと XML との変換にこの方法を使用することの利点は、XML パーサーに備えられている以上の機能が CICS によって提供されることです。 CICS は XML を解釈して、アプリケーション・データのレコード・ベースの変換を実行できます。したがって、このアプローチを使用することで、XML を処理するアプリケーションをより簡単かつ迅速に作成できます。

付属のユーティリティーである CICS XML アシスタントは、アプリケーションのバイナリー・データを XML に変換したり、XML をアプリケーションのバイナリー・データに変換したりするのに必要な成果物を作成するのに役立ちます。XML アシスタントは成果物をバンドル・ディレクトリーか、z/OS UNIX の指定した別の場所に作成できます。

手順

1. XML アシスタントを使用してマッピングを作成します。
2. CICS で XMLTRANSFORM リソースを作成して、マッピングを使用可能にします。
3. アプリケーション・プログラムを作成または更新して、**TRANSFORM** API コマンドを使用します。 そのアプリケーションはチャネル・ベースのインターフェースを使用する必要があります。
4. アプリケーションを実行して、変換が意図したとおりに行われるかどうかをテストします。 妥当性検査をオンにして、CICS がデータを正しく変換するかどうかを確認することができます。

次のタスク

これらのステップの詳細については、次のトピックで説明します。

CICS XML 支援機能

CICS XML 支援機能は、 XML と高水準言語構造との間の相互変換を支援する一連のバッチ・ユーティリティーです。 この支援機能により、 XML 処理を実行するアプリケーションを最小限のプログラミングで迅速にデプロイすることが可能になります。

CICS 用の XML 支援機能を使用すると、 XML の解析または構築のために作成する必要のあるコードの量を軽減できます。 CICS では、XML フラグメントとアプリケーション・プログラムのデータ構造との間でデータが相互に変換されます。

XML 支援機能では、単純な言語構造からの XML スキーマの作成や既存の XML スキーマからの言語構造の作成が可能であり、COBOL、C/C++、および PL/I をサポートしています。また、 CICS が実行時に XML データをバイナリー・アプリケーション・データに自動的に変換する (または逆の変換を行う) ために使用するメタデータも生成されます。このメタデータは XML バインディングで定義され、z/OS UNIX 上に保管されます。 XML バインディング用のスキーマは、z/OS UNIX 上の `/usr/lpp/cicsts/cicsts52/schemas/xmltransform/` ディレクトリーにあります。

CICS XML 支援機能は以下の 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。

DFHLS2SC
: このユーティリティーは言語構造から XML スキーマおよびバインディングを生成します。

DFHSC2LS
: 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 XML バインディングおよび言語構造を生成します。 WSDL 文書または XML スキーマを入力データとして使用できます。

hlq`.XDFHINST` ライブラリーには、両方のプログラムを実行するための JCL プロシージャーがあります (*hlq* は CICS インストールの高位修飾子です)。

DFHLS2SC: 高水準言語から XML スキーマへの変換

カタログ式プロシージャーの DFHLS2SC は、XML スキーマと XML バインディング・ファイルを高水準言語構造から生成します。 XML の構文解析または作成が可能な CICS プログラムを作成する場合は、DFHLS2SC を使用してください。

DFHLS2SC 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 (DFHLS2SC) を指定します。

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・セットまたは区分データ・セットのメンバーで定義することもできます。

シンボリック・パラメーター

DFHLS2SC では以下のシンボリック・パラメーターが定義されています。

JAVADIR = *path*
: DFHLS2SC 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が `/usr/lpp/` に付加されることにより、完全なパス名 `/usr/lpp/`*path* が形成されます。

通常、このパラメーターは指定しません。 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。

PATHPREF = *prefix*
他のパラメーターで使用される z/OS UNIX ディレクトリー・パスを拡張する、オプションの接頭部を指定します。 デフォルトは空ストリングです。

通常、このパラメーターは指定しません。 デフォルト値は、**PATHPREF** 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。

SERVICE = *value*
このパラメーターは、IBM サポートに指示された場合にのみ使用します。

TMPDIR = *tmpdir*
DFHLS2SC が z/OS UNIX 内で一時ワークスペースとして使用する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・ファイルの名前を構成するときに DFHLS2SC によって使用される接頭部を指定します。

デフォルト値は `SC2WS` です。

USSDIR = *path*
z/OS UNIX ファイル・システム内の 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/` に付加されて、`/usr/lpp/cicsts/`*path* の完全なパス名が形成されます。

通常、このパラメーターは指定しません。 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。

一時ワークスペース

DFHLS2SC により、実行時に以下の 3 つの一時ファイルが作成されます。

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名 (**TMPDIR** および **TMPFILE** が指定されていない場合) は、次のとおりです。

```
/tmp/LS2SC.in
/tmp/LS2SC.out
/tmp/LS2SC.err
```

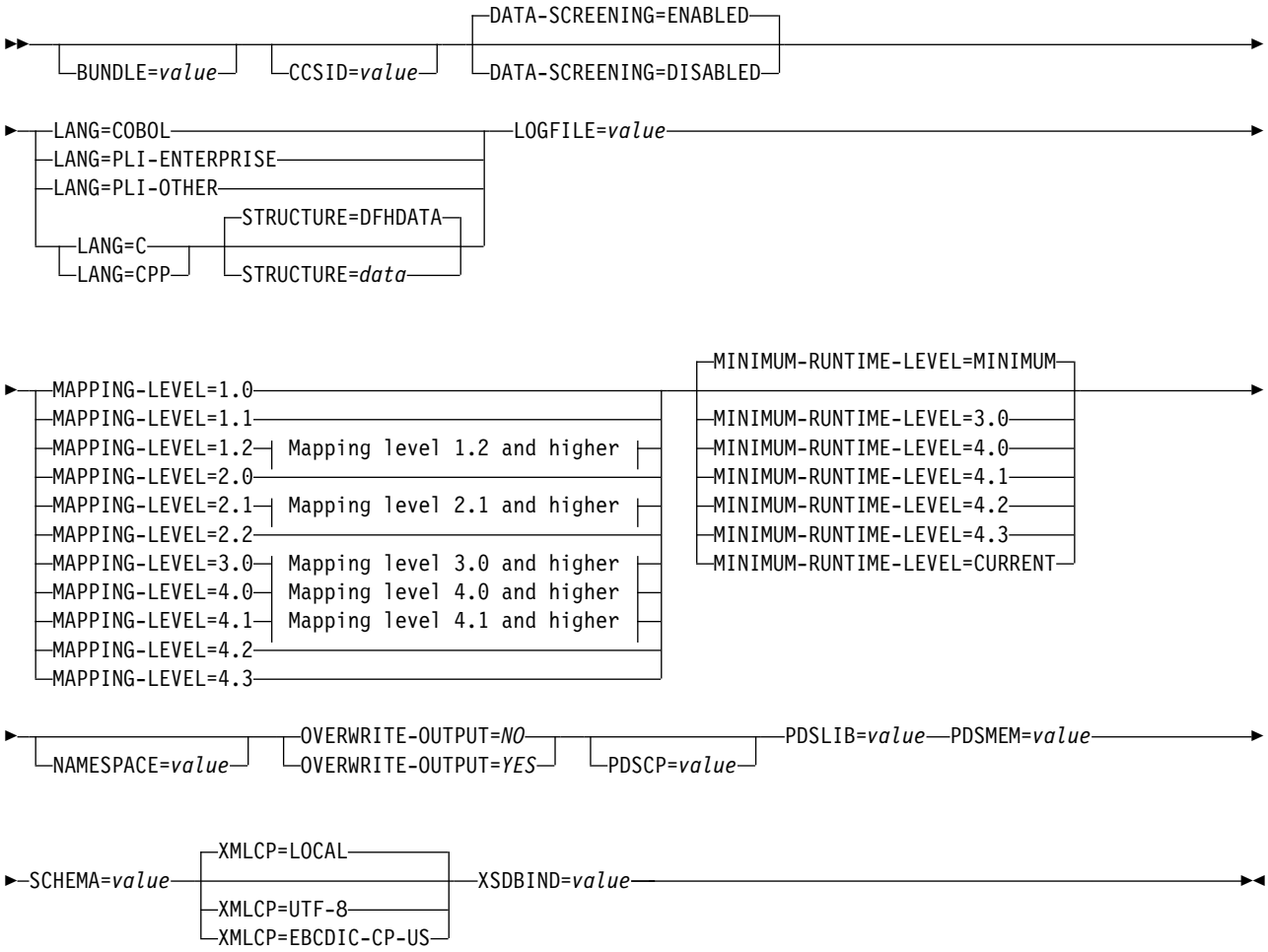
重要: DFHLS2SC は、z/OS UNIX ファイルまたはデータ・セット・メンバーへのアクセスをロックしません。したがって、DFHLS2SC の複数のインスタンスが同時に実行され、同じ一時ワークスペース・ファイルを使用する場合、あるジョブが

ワークスペース・ファイルを使用しているときに、他のジョブがそれらのファイルを上書きするのを防止することはできないため、予測不能な障害が生じます。

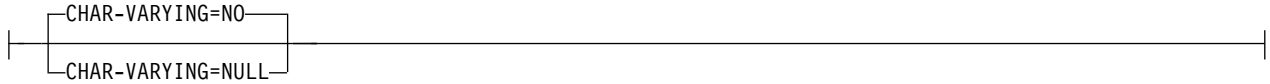
そのため、このような状況を回避する命名規則と操作手順を考案することをお勧めします。例えば、システム・シンボリック・パラメーターの **SYSUID** を使用して、個々のユーザーにとって固有のワークスペース・ファイル名を生成することができます。

これらの一時ファイルは、ジョブが終了する前に削除されます。

DFHLS2SC の入力パラメーター

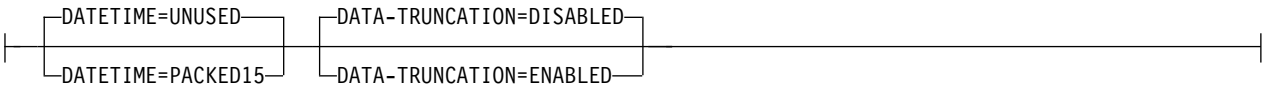


マッピング・レベル **1.2** 以上の場合:

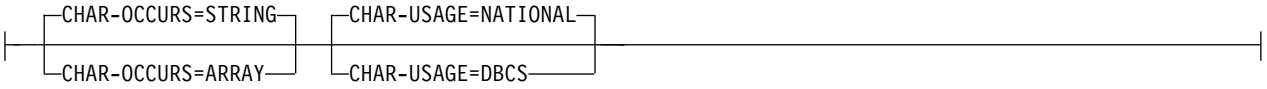


マッピング・レベル **2.1** 以上の場合:

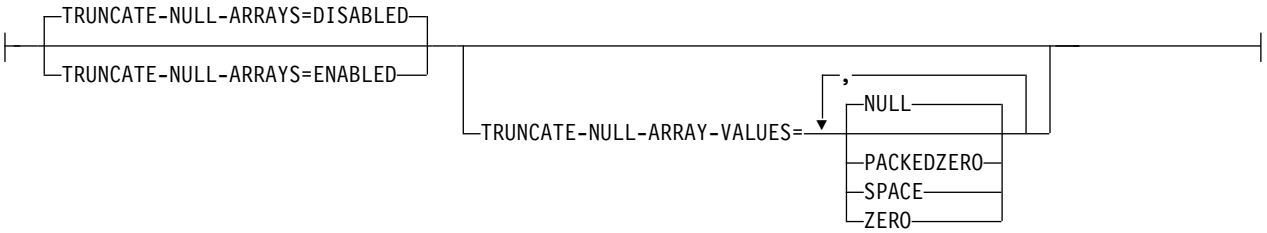
マッピング・レベル **3.0** 以上**:**



マッピング・レベル **4.0** 以上の場合**:**



マッピング・レベル **4.1** 以上の場合**:**



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 (および使用する場合は、その継続文字) は、72 桁を超えて拡張することはできません。73 桁から 80 桁の間はブランクにする必要があります。
- 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。 アスタリスクの前のすべての部分 (スペースを含む) は、パラメーターの一部とみなされます。以下に例を示します。

  ```
  XSDBIND=/path/xsdbinddir*
        /app1
  ```

  このコードは、次のコードと同じ意味になります。

  ```
  XSDBIND=/path/xsdbinddir/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。
- 行の末尾の文字の位置にコンマがある場合、これはオプションの行分離文字であり、無視されます。

パラメーターの記述

BUNDLE = *value*

z/OS UNIX 上のバンドル・ディレクトリーのパスと名前を指定します。この値を指定すると、XML 支援機能によって、XSD バインディングを組み込んだバンドルが生成されます。 このパラメーターのパス情報は、**XSDBIND** パラメーター上のパス情報を指定変更します。

ディレクトリー名の代わりにアーカイブ・ファイルを指定することもできます。XML 支援機能では `.zip` および `.jar` アーカイブがサポートされています。ただし、BUNDLE リソースをインストールする前にアーカイブを圧縮解除する必要があります。

このパラメーターを指定しない場合は、CICS によって、**XSDBIND** パラメーターで指定されている場所に XML スキーマとバインディングが配置されます。

CCSID = *value*

アプリケーション・データ構造に文字データをエンコードするために、実行時に使用される CCSID を指定します。このパラメーターの値は、**LOCALCCSID** システム初期設定パラメーターの値を指定変更します。 *value* は、Java および z/OS 変換サービスによってサポートされている EBCDIC CCSID である必要があります。 このパラメーターを指定しない場合、アプリケーション・データ構造は、システム初期設定パラメーターで指定された CCSID を使用してエンコードされます。

このパラメーターは任意のマッピング・レベルで使用することができます。

CHAR-VARYING = { **NO** | **NULL** | **COLLAPSE** | **BINARY** }

マッピング・レベルが 1.2 以上のとき、言語構造内の文字フィールドがマップされる方法を指定します。 COBOL の文字フィールドは、タイプ X のピクチャー節 (`PIC(X) 10` など) です。 C/C++ の文字フィールドは文字配列です。このパラメーターは、Enterprise および他の PL/I 言語構造には適用されません。 以下のオプションを選択できます。

NO
: 文字フィールドは `xsd:string` にマップされ、固定長のフィールドとして処理されます。 データの最大長はフィールドの長さと同じです。NO は、マッピング・レベル 2.0 以前での、COBOL および PL/I の **CHAR-VARYING** パラメーターのデフォルト値です。

NULL
: 文字フィールドは `xsd:string` にマップされ、ヌル終了ストリングとして処理されます。 CICS は、XML スキーマからの変換時に、終了を表すヌル文字を追加します。文字ストリングの最大長は、言語構造に示される長さより 1 文字少ない長さとして計算されます。 NULL は、C/C++ での **CHAR-VARYING** パラメーターのデフォルト値です。

COLLAPSE
: 文字フィールドは `xsd:string` にマップされます。フィールド内の後続空白と埋め込み空白は XML スキーマに含まれません。例えば、`<space>AB<space><space><space>C<space>` は `AB<space>C` になります。 COLLAPSE は、マッピング・レベル 2.1 以上での、COBOL および PL/I の **CHAR-VARYING** パラメーターのデフォルト *value* です。

BINARY

文字フィールドは xsd:base64binary データ型としてマップされ、固定長フィールドとして処理されます。 **CHAR-VARYING** パラメーター上の BINARY *value* は、マッピング・レベル 2.1 以上でのみ使用可能です。

CHAR-OCCURS = { **STRING** | **ARRAY** }

マッピング・レベル 4.0 以上の場合に、言語構造内の文字配列をマップする方法を指定します。例えば PIC X OCCURS 20 となります。このパラメーターは COBOL 言語でのみ使用されます。

ARRAY

文字配列は XML 配列にマップされます。つまり、それぞれの文字が個別の XML エレメントとしてマップされます。マッピング・レベル 3.0 以前でも、このように動作します。

STRING

文字配列は XML ストリングにマップされます。つまり、COBOL 配列全体が 1 つの XML エレメントとしてマップされます。

CHAR-USAGE = { **NATIONAL** | **DBCS** }

COBOL では、各国語データ型 PIC N を UTF-16 または DBCS データ用に使用できます。この設定は NSYMBOL コンパイラー・オプションによって制御されます。データが適切に扱われるようにするには、アシスタントの **CHAR-USAGE** パラメーターを NSYMBOL コンパイラー・オプションと同じ値に設定する必要があります。 UTF-16 を使用する場合には、通常、これは CHAR-USAGE=NATIONAL に設定されます。

DBCS

PIC (*n*) フィールドからのデータは、DBCS でエンコードされたデータとして扱われます。

NATIONAL

PIC (*n*) フィールドからのデータは、UTF-16 でエンコードされたデータとして扱われます。

DATA-SCREENING = { **ENABLED** | **DISABLED** }

アプリケーション提供のデータのエラーを検査するかどうかを指定します。

ENABLED

言語構造と矛盾するアプリケーション提供のランタイム・データは、エラーとして扱われ、メッセージ DFHPI1010 が発行されます。エラー応答がアプリケーションに 返されます。

DISABLED

言語構造と矛盾するアプリケーション提供のランタイム・データの値は、 デフォルト値で置き換えられます。例えば、数値フィールド内のゼロは誤った値を置き換えます。メッセージ DFHPI1010 は発行されず、通常の応答がアプリケーションに返されます。この機能を使用して、 初期設定されていない出力フィールドから生成される INVALID_PACKED_DEC および INVALID_ZONED_DEC エラー応答を回避できます。

DATA-TRUNCATION = { **DISABLED** | **ENABLED** }

固定長フィールド構造で可変長データを許容するかどうかを指定します。

DISABLED
データが CICS が予期する固定長より短い場合に、CICS は切り捨てられたデータを拒否してエラー・メッセージを発行します。

ENABLED
データが CICS が予期する固定長より短い場合に、CICS は切り捨てられたデータを許容して、欠落データをヌル値として処理します。

DATETIME = { **UNUSED** | **PACKED15** }
高水準言語構造内で ABSTIME になる可能性のあるフィールドをタイム・スタンプとしてマップするかどうかを指定します。

PACKED15
長さ 15 (8 バイト) のパック 10 進数フィールドは CICS の ABSTIME フィールドとして扱われ、タイム・スタンプとしてマップされます。

UNUSED
長さ 15 (8 バイト) のパック 10 進数フィールドは、タイム・スタンプとしては扱われません。

このパラメーターは、マッピング・レベル 3.0 で設定できます。

LANG = **COBOL**|**PLI-ENTERPRISE**|**PLI-OTHER**|**C**|**CPP**
高水準言語構造のプログラミング言語を指定します。

COBOL
COBOL

PLI-ENTERPRISE
Enterprise PL/I

PLI-OTHER
Enterprise PL/I 以外の PL/I のレベル

C C

CPP C++

LOGFILE = *value*
DFHLS2SC によって自身のアクティビティー・ログが書き込まれ、情報がトレースされるファイルの、完全修飾された z/OS UNIX 名。 このファイルが存在しない場合、DFHLS2SC はこのファイルを作成しますが、ディレクトリー構造は作成しません。

通常、このファイルを使用することはありませんが、DFHLS2SC で問題が発生した場合は、IBM サービス組織に要求される場合があります。

| **MAPPING-LEVEL** = **{ 1.0 | 1.1 | 1.2 | 2.0 | 2.1 | 2.2 | 3.0 | 4.0 | 4.1 | 4.2 | 4.3 }**
XML バインディングおよび言語構造の生成時に支援機能で使用するマッピング・レベルを指定します。 使用可能な最新のマッピング・レベルを使用することをお勧めします。 Atom フィードの XML バインディングを作成している場合は、マッピング・レベル 3.0 を使用する必要があります。

| **MINIMUM-RUNTIME-LEVEL** = **{ MINIMUM | 3.0 | 4.0 | 4.1 | 4.2 | 4.3 | CURRENT }**
Web サービス・バインディング・ファイルを配置できる最小の CICS 実行時環

境を指定します。指定した他のパラメーターと一致しないレベルを選択すると、エラー・メッセージが送信されます。 以下のオプションを選択できます。

MINIMUM
選択したパラメーターを前提として、最低限可能な CICS 実行時レベルが自動的に割り振られます。

3.0 CICS XML 支援機能を使用して、高機能のデータ・マッピングを利用する場合は、ランタイム・レベル 3.0 以上を指定します。

4.0 生成された Web サービス・バインディング・ファイルは、CICS TS 5.2 以降の領域に正常にデプロイされます。この実行時レベルでは、**MAPPING-LEVEL** パラメーターにマッピング・レベル 4.0 以前を使用できます。このレベルでは任意のオプション・パラメーターを使用できます。

4.1 生成された Web サービス・バインディング・ファイルは、CICS TS 5.2 以降の領域に正常にデプロイされます。この実行時レベルでは、**MAPPING-LEVEL** パラメーターには、マッピング・レベル 4.1 以前を使用することができます。

4.2 生成された Web サービス・バインディング・ファイルは、CICS TS V5.4 以降の領域に正常にデプロイされます。この実行時レベルでは、**MAPPING-LEVEL** パラメーターには、マッピング・レベル 4.2 以前を使用することができます。

4.3 生成された Web サービス・バインディング・ファイルは、CICS TS 5.4 以降の領域に正常にデプロイされます。この実行時レベルでは、**MAPPING-LEVEL** パラメーターには、マッピング・レベル 4.3 以前を使用することができます。

CURRENT
生成された Web サービス・バインディング・ファイルは、Web サービス・バインディング・ファイルの生成に使用するものと同じ実行時レベルで、CICS 領域に正常に配置されます。

NAMESPACE = *value*
生成された XML スキーマで使用する CICS の名前空間を指定します。 Atom フィードの場合、CICS は Atom 名前空間とともに Atom フィードでこの名前空間を提供します。

このパラメーターを指定しない場合、CICS では名前空間が自動的に生成されます。

OVERWRITE-OUTPUT = **NO** | **YES**
ファイル・システム上の既存の CICS BUNDLE が上書き可能かどうかを制御します。

NO 既存 BUNDLE は何も置き換えられません。 既存 BUNDLE が検出された場合、DFHLS2SC はエラー・メッセージ DFHPI9689E を発行し強制終了します。

YES どの既存 BUNDLE も置き換えられます。 既存 BUNDLE が検出された場合、メッセージ DFHPI9683W が発行され、ファイルが置き換えられたことを通知します。

PDSCP = *value*
区分データ・セット・メンバーで使用されるコード・ページを指定します。ここで、*value* は CCSID 番号または Java コード・ページ番号です。 このパラメーターを指定しない場合、z/OS UNIX システム・サービス・コード・ページが使用されます。例えば、PDSCP=037 を指定することができます。

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。

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。

PDSMEM = *value*
処理される高水準言語構造を含む区分データ・セット・メンバーの名前を指定します。

SCHEMA = *value*
XML スキーマが書き込まれるファイルの、完全修飾された z/OS UNIX 名。XML スキーマは WSDL 2.0 仕様に準拠しています。 このファイルが存在しない場合、DFHLS2SC はこのファイルを作成しますが、ディレクトリー構造は作成しません。

STRUCTURE ={**DFHDATA** | *data*}
C および C++ での、最上位データ構造の名前。デフォルトは DFHDATA です。

TRUNCATE-NULL-ARRAYS = { **DISABLED** | **ENABLED** }
マッピング・レベル 4.1 以上で構造化配列を処理する方法を指定します。この機能を有効にすると、CICS は配列に含まれる空のレコードを認識しようと試みます (空のレコードの認識について詳しくは、TRUNCATE-NULL-ARRAY-VALUES を参照)。空の配列レコードが 5 個連続して検出された場合、その配列は XML/JSON の生成時に最初の空のレコードの箇所で切り捨てられます。この機能は、内容が構造化されている配列に対してのみ有効にされます。単純なプリミティブ・フィールドからなる配列には、切り捨ては適用されません。配列の切り捨てにより、JSON/XML 内のデータ表現が簡潔になりますが、リスクがないわけではありません。5 個の連続したデータ・レコードが、(おそらく、正当な低値を含んでいるために) 初期化されていないストレージとして誤って認識された場合、データ損失が生じるおそれがあります。TRUNCATE-NULL-ARRAYS が有効にされていて、TRUNCATE-NULL-ARRAY-VALUES が設定されていない場合は、TRUNCATE-NULL-ARRAY-VALUES のデフォルト値が使用されます。

TRUNCATE-NULL-ARRAY-VALUES = { **NULL** | **PACKEDZERO**|**SPACE** | **ZERO** }
マッピング・レベル 4.1 以上で、TRUNCATE-NULL-ARRAYS の処理で空として扱う値を指定します。デフォルトでは、ヌル値 (0x00、または小さい値) が空として扱われます。構造化された配列のレコードに含まれるすべてのストレージ・バイトにヌルが含まれている場合、レコード全体が空であると見なされます。1 つ以上の NULL、PACKEDZERO、SPACE、および ZERO 値をコンマ区切りリストに指定できます。

NULL
ヌル文字 (0x00) を暗黙指定します。

PACKEDZERO
正符号付きパック 10 進数ゼロ (0x0C)、負符号付きパック 10 進数ゼロ (0x0D)、または符号なしパック 10 進数ゼロ (0x0F) を暗黙指定します。

SPACE
SBCS EBCDIC スペース (0x40) を暗黙指定します。

ZERO
符号なしのゾーン 10 進数ゼロ (0xF0) を暗黙指定します。

構造化配列レコードに、選択したバイトの組み合わせとの一致が含まれている場合、レコード全体が空として識別されます。

TRUNCATE-NULL-ARRAY-VALUES に値が定義されている場合、TRUNCATE-NULL-ARRAYS が有効に設定されている必要があります。

XSDBIND = *value*
XSD バインディングの完全修飾された z/OS UNIX 名。 このファイルが存在しない場合、DFHLS2SC はこのファイルを作成しますが、ディレクトリー構造は作成しません。 BUNDLE パラメーターが指定された場合、パスを除外します。ファイル拡張子は .xsdbind です。

DFHSC2LS: XML スキーマから高水準言語への変換

カタログ式プロシージャーの DFHSC2LS では、高水準言語データ構造および XML バインディングが XML スキーマまたは WSDL 文書から生成されます。 XML の構文解析または作成が可能な CICS プログラムを作成する場合は、DFHSC2LS を使用してください。 このトピックでは、DFHSC2LS のジョブ制御ステートメント、シンボリック・パラメーター、入力パラメーター、およびその説明をリストします。

DFHSC2LS 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 (DFHSC2LS) を指定します。

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・セットまたは区分データ・セットのメンバーで定義することもできます。

シンボリック・パラメーター

DFHSC2LS では以下のシンボリック・パラメーターが定義されています。

JAVADIR = *path*
DFHSC2LS 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が /usr/lpp/ に付加されることにより、完全なパス名 /usr/lpp/*path* が形成されます。

通常、このパラメーターは指定しません。 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。

PATHPREF = *prefix*
他のパラメーターで使用される z/OS UNIX ディレクトリー・パスを拡張する、オプションの接頭部を指定します。デフォルトは空ストリングです。

通常、このパラメーターは指定しません。 デフォルト値は、**PATHPREF** 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*
DFHSC2LS が一時ワークスペースとして使用する z/OS UNIX 内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*
DFHSC2LS が一時ワークスペース・ファイルの名前を構成するために使用する接頭部を指定します。

デフォルト値は SC2LS です。

USSDIR = *path*
z/OS UNIX ファイル・システム内の 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/ に付加されて、/usr/lpp/cicsts/*path* の完全なパス名が形成されます。

通常、このパラメーターは指定しません。 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。

SERVICE = *value*
このパラメーターは、IBM サポートに指示された場合にのみ使用します。

一時ワークスペース

DFHSC2LS により、実行時に以下の 3 つの一時ファイルが作成されます。

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名 (**TMPDIR** および **TMPFILE** が指定されていない場合) は、次のとおりです。

/tmp/SC2LS.in

/tmp/SC2LS.out

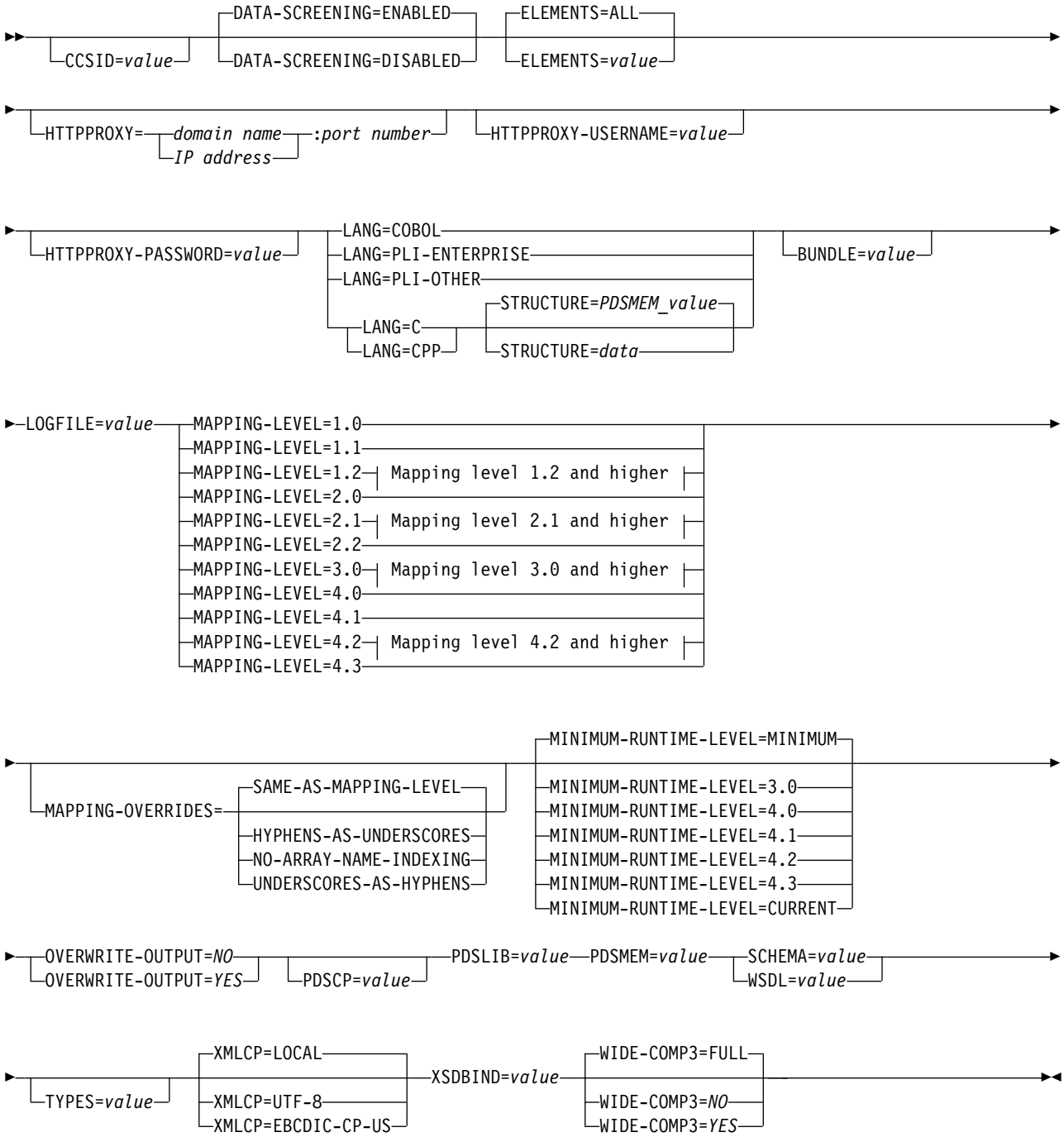
/tmp/SC2LS.err

重要: DFHSC2LS は、z/OS UNIX ファイルまたはデータ・セット・メンバーへのアクセスをロックしません。したがって、DFHSC2LS の複数のインスタンスが同時に実行され、同じ一時ワークスペース・ファイルを使用する場合、あるジョブがワークスペース・ファイルを使用しているときに、他のジョブがそれらのファイルを上書きするのを防止することができないため、予測不能な障害が生じます。

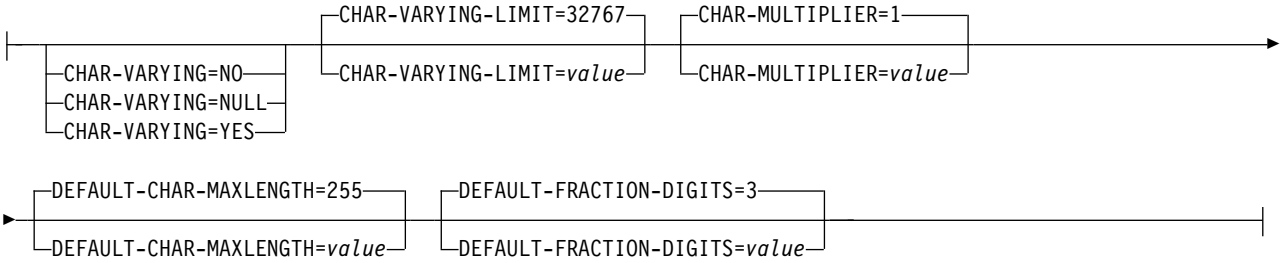
そのため、このような状況を回避する命名規則と操作手順を考案することをお勧めします。例えば、システム・シンボリック・パラメーターの **SYSUID** を使用して、個々のユーザーにとって固有のワークスペース・ファイル名を生成することができます。

これらの一時ファイルは、ジョブが終了する前に削除されます。

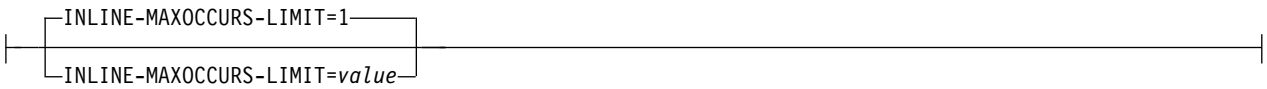
DFHSC2LS の入力パラメーター



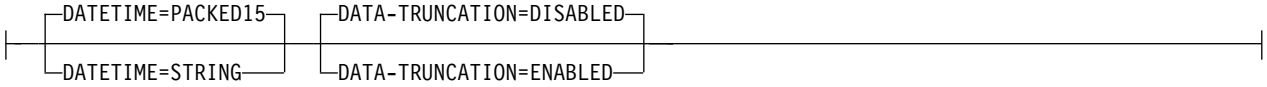
マッピング・レベル **1.2** 以上の場合:



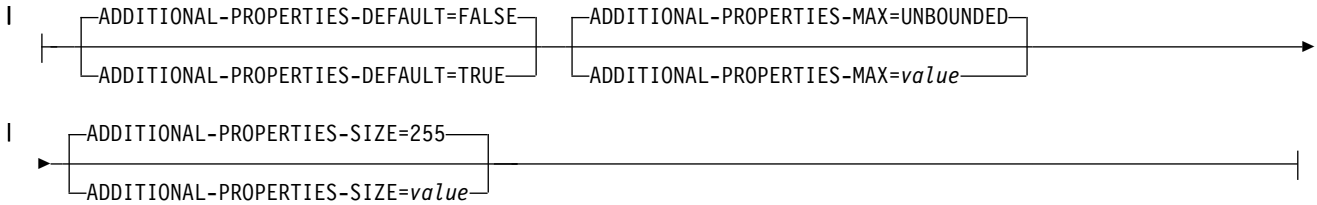
マッピング・レベル **2.1** 以上の場合:



マッピング・レベル **3.0** 以上:



マッピング・レベル **4.2** 以上の場合:



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 (および使用する場合は、その継続文字) は、72 桁を超えて拡張することはできません。73 桁から 80 桁の間はブランクにする必要があります。
- 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。 アスタリスクの前のすべての部分 (スペースを含む) は、パラメーターの一部とみなされます。以下に例を示します。

  ```
  XSDBIND=xsdbinddir*
        /app1
  ```

  このコードは、次のコードと同じ意味になります。

  ```
  XSDBIND=xsdbinddir/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

ADDITIONAL-PROPERTIES-DEFAULT = { **true** | **false** }
追加プロパティーのサポートを明示的に宣言していない JSON スキーマ・オブジェクトが、それらのプロパティーをサポートしていると解釈されるかどうかを示します。追加の JSON プロパティーは、JSON スキーマ内で事前に定義されていない JSON オブジェクト内のプロパティーです。これらのプロパティーは、通常、予期しない追加データとしてデータ変換メカニズムによって拒否されます。**ADDITIONAL-PROPERTIES-DEFAULT** が TRUE に設定されている場合、または JSON スキーマによってオブジェクトに対して additionalProperties:true が明示的に設定されている場合、生成されるコピーブックにこれらの値を保持するためのスペースが割り当てられます。アプリケーションは、コピーブック内の関連するフィールドを使用して、それらの値と対話できます。

このパラメーターは、マッピング・レベル 4.2 以上で使用できます。

ADDITIONAL-PROPERTIES-MAX = { **0-20** | **UNBOUNDED** }
追加プロパティーをサポートする JSON オブジェクトに対してサポートされるこれらのプロパティーの数を示します。**ADDITIONAL-PROPERTIES-DEFAULT** を参照してください。生成されるコピーブックには、追加プロパティーのアドレッシングに適した構造が含まれます。デフォルトでは、サポートされるプロパティーの数に最大制約はありません。コピーブックは、制約のない配列と同様の方法で生成され、コンテナーを使用します。このパラメーターを **INLINE-MAXOCCURS-LIMIT** パラメーターと組み合わせて使用可能な最大制約を適用し、最大数のプロパティー用に固定長の配列を割り当てることができます。これにより、コンテナーは不用になります。

このパラメーターは、マッピング・レベル 4.2 以上で使用できます。

ADDITIONAL-PROPERTIES-SIZE = { **16-32767** | **255** }
各 JSON 追加プロパティーの最大サイズを示します。JSON オブジェクトが **ADDITIONAL-PROPERTIES-DEFAULT** によって定義された追加プロパティーをサポートする場合、生成されるコピーブックには、**ADDITIONAL-PROPERTIES-MAX** で指定された数までプロパティーをサポートするバインディングが設定されます。デフォルトでは、各追加プロパティーでサポートされる最大値は 255 文字です。そのサイズのフィールドは、生成されるコピーブックに生成されます。このサイズは、**ADDITIONAL-PROPERTIES-SIZE** パラメーターを設定することでカスタマイズできます。例えば、JSON オブジェクトは、以下のプロパティーを含むように処理されます。

"example": { "notes": "this extra property was not defined in the JSON Schema" }

追加プロパティーをサポートするようにコピーブックが生成されている場合は、その値全体がアプリケーションに渡されて処理されます。この値は、プロパティーのキーの前の先頭引用符で始まり、プロパティーの値の末尾の右中括弧で終わります。この例では約 100 文字です。**ADDITIONAL-PROPERTIES-SIZE** に使用する値は、考えられる最大の値を保持できる大きさにする必要があります。処理される値に対して割り当てられたバッファーが小さすぎると、エラー応答が生成されます。

このパラメーターは、マッピング・レベル 4.2 以上で使用できます。

CCSID = *value*
アプリケーション・データ構造に文字データをエンコードするために、実行時に使用される CCSID を指定します。このパラメーターの値は、**LOCALCCSID** システム初期設定パラメーターの値を指定変更します。 *value* は、Java および z/OS 変換サービスによってサポートされている EBCDIC CCSID である必要があります (z/OS Unicode Services ユーザーズ・ガイドおよび解説書を参照)。このパラメーターを指定しない場合、アプリケーション・データ構造は、システム初期設定パラメーターで指定された CCSID を使用してエンコードされます。

このパラメーターは任意のマッピング・レベルで使用できます。

CHAR-MULTIPLIER = { **1** | *value* }
マッピング・レベルが 1.2 以降のとき、各文字に許可されるバイト数を指定します。このパラメーターの *value* は、1 から 2,147,483,647 の範囲の正整数です。 非数字に基づくマッピングはすべて、この乗数の対象です。バイナリー、数値、ゾーンおよびパック 10 進数フィールドは、この乗数の対象ではありません。

このパラメーターが役に立つのは、例えば、実行時にすべての 2 バイト文字の周囲に潜在的なシフトアウト文字とシフトイン文字のスペースを許可するために、乗数 3 を選択できる DBCS 文字の使用を計画している場合などです。

(UTF-16 を示す) **CCSID=1200** を設定する場合、**CHAR-MULTIPLIER** の有効値は 2 または 4 のみです。 UTF-16 を使用する場合のデフォルト値は 2 です。1 つの UTF-16 エンコード・ユニットを必要とする文字がアプリケーション・データに含まれると予期される場合は、**CHAR-MULTIPLIER=2** を使用してください。 2 つの UTF-16 エンコード・ユニットを必要とする文字がアプリケーション・データに含まれると予期される場合は、**CHAR-MULTIPLIER=4** を使用してください。

注**:** **CHAR-MULTIPLIER** を 1 に設定した場合、DBCS 文字は使用不可にはならず、2 に設定した場合、UTF-16 代理ペアは使用不可にはなりません。ただし、ワイド文字が定期的に使用される場合、いくつかの有効値は、割り振られたフィールドに入らなくなります。より大きな **CHAR-MULTIPLIER** 値を使用すると、XML で有効な文字数よりも多くの文字を、割り振られたフィールドに格納できます。該当する範囲制限を守るように注意する必要があります。

CHAR-VARYING = { **NO** | **NULL** | **YES** }
マッピング・レベルが 1.2 以上のとき、可変長文字データがマップされる方法を指定します。 可変長バイナリー・データ・タイプは、常にコンテナーまたは可変構造のいずれかにマップされます。 このパラメーターを指定しない場合、デフォルトのマッピングは、指定された言語によって決まります。 以下のオプションを選択できます。

NO 可変長文字データは固定長ストリングとしてマップされます。

NULL
可変長文字データはヌル終了ストリングにマップされます。

YES PL/I の場合、可変長文字データは CHAR VARYING データ型にマップされます。COBOL、C、および C++ 言語の場合、可変長文字データは、 2 つの関連エレメント (データ長およびデータ) で構成される同等の表現にマップされます。

CHAR-VARYING-LIMIT = { **32767** | *value* }
マッピング・レベルが 1.2 以上のとき、言語構造にマップされるバイナリー・データおよび可変長文字データの最大サイズを指定します。 文字データまたはバイナリー・データが、このパラメーターで指定される値より大きい場合は、コンテナーにマップされ、コンテナー名が生成された言語構造で使用されます。 *value* は 0 からデフォルトの 32 767 バイトの範囲で指定できます。

DATA-SCREENING = { **ENABLED** | **DISABLED** }
アプリケーション提供のデータのエラーを検査するかどうかを指定します。

ENABLED
言語構造と矛盾するアプリケーション提供のランタイム・データは、エラーとして扱われ、メッセージ DFHPI1010 が発行されます。エラー応答がアプリケーションに 返されます。

DISABLED
言語構造と矛盾するアプリケーション提供のランタイム・データの値は、 デフォルト値で置き換えられます。例えば、数値フィールド内のゼロは誤った値を置き換えます。メッセージ DFHPI1010 は発行されず、通常の応答がアプリケーションに返されます。この機能を使用して、 初期設定されていない出力フィールドから生成される INVALID_PACKED_DEC および INVALID_ZONED_DEC エラー応答を回避できます。

DATA-TRUNCATION = { **DISABLED** | **ENABLED** }
固定長フィールド構造で可変長データを許容するかどうかを指定します。

DISABLED
データが CICS が予期する固定長より短い場合に、CICS は切り捨てられたデータを拒否してエラー・メッセージを発行します。

ENABLED
データが CICS が予期する固定長より短い場合に、CICS は切り捨てられたデータを許容して、欠落データをヌル値として処理します。

DATETIME = { **PACKED15** | **STRING** }
xsd:dateTime フィールドが CICS ABSTIME データ形式またはテキストにマップされることを指定します。

PACKED15
xsd:dateTime フィールドは CICS ABSTIME 形式にマップされます。

STRING
xsd:dateTime フィールドはテキストにマップされます。 このマッピングはこれまでのすべてのマッピング・レベルと同じです。

このパラメーターはマッピング・レベル 3.0 で使用できます。

DEFAULT-CHAR-MAXLENGTH = { **255** | *value* }
マッピング・レベルが 1.2 以上で、XML スキーマ文書または WSDL 文書で長さが指定されていない場合に、マッピングする文字の文字データのデフォルトの配列長を指定します。 このパラメーターの *value* は、1 から 2 147 483 647 の範囲の正整数で指定できます。

DEFAULT-FRACTION-DIGITS = **3** | *value*
XML 10 進スキーマ・タイプで使用するデフォルトの小数桁数を指定します。 デフォルト値は 3 です。 COBOL の場合、有効な範囲は 0 から 17、または パラメーター **WIDE-COMP3** が使用されている場合、0 から 30 です。 C または PL/I の場合、有効な範囲は 0 から 30 までです。

ELEMENTS = { **ALL** | *value* }
有効にするグローバル・エレメントのローカル名のリストを定義します。 デフォルト値 ALL は、すべてのグローバル・エレメントが有効であることを示します。

HTTPPROXY = { *domain name* | *IP address* } : *port number*
XML スキーマまたは WSDL 文書に、インターネット上にある他の XML スキーマまたは WSDL ファイルへの参照が含まれていて、DFHSC2LS を実行しているシステムがインターネットへのアクセスにプロキシー・サーバーを使用している場合は、そのプロキシー・サーバーのドメイン名か IP アドレス、およびポート番号を指定します。 以下に例を示します。

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

それ以外の場合、このパラメーターは不要です。

HTTPPROXY-USERNAME = *value*
DFHSC2LS を実行しているシステムが HTTP 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し、HTTP プロキシー・サーバーが基本認証を使用している場合に、**HTTPPROXY-PASSWORD** と共に使用する HTTP プロキシー・ユーザー名を指定します。 **HTTPPROXY** も指定した場合にのみ、このパラメーターを使用できます。

HTTPPROXY-PASSWORD = *value*
DFHSC2LS を実行しているシステムが HTTP 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し、HTTP プロキシー・サーバーが基本認証を使用している場合に、**HTTPPROXY-USERNAME** と共に使用する HTTP プロキシー・パスワードを指定します。 **HTTPPROXY** も指定した場合にのみ、このパラメーターを使用できます。

INLINE-MAXOCCURS-LIMIT = { **1** | *value* }
インラインの可変反復内容を XML 属性の maxOccurs 属性に基づいて使用するかどうかを指定します。

INLINE-MAXOCCURS-LIMIT パラメーターは、マッピング・レベル 2.1 以降でのみ使用できます。 **INLINE-MAXOCCURS-LIMIT** の *value* は、0 から 32 767 の範囲の正整数で指定できます。 値が 0 の場合は、インライン・マッピングが使用されないことを示します。 値が 1 の場合は、オプションのエレメントがインラインでマップされることを示します。 maxOccurs 属性の *value* が **INLINE-MAXOCCURS-LIMIT** の *value* より大きい場合は、コンテナー・ベースのマッピングが使用されます。それ以外の場合はインライン・マッピングが使用されます。

可変の繰り返しリストをインラインでマップするかどうか決定する際には、繰り返されるデータの単一項目の長さを考慮してください。長いインスタンスがほとんどない場合は、コンテナー・ベースのマッピングの方が望ましい方法です。短いインスタンスが多数ある場合は、インライン・マッピングの方が望ましい可能性があります。

LANG = COBOL|PLI-ENTERPRISE|PLI-OTHER|C|CPP
高水準言語構造のプログラミング言語を指定します。

COBOL
COBOL

PLI-ENTERPRISE
Enterprise PL/I

PLI-OTHER
Enterprise PL/I 以外の PL/I のレベル

C C

CPP C++

LOGFILE = *value*
DFHSC2LS 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、完全修飾された z/OS UNIX 名。 DFHSC2LS はこのファイルを作成しますが、存在していないディレクトリー構造を作成することはありません。

通常、このファイルを使用することはありませんが、DFHSC2LS で問題が発生した場合は、IBM サービス組織によって使用が指示される場合があります。

MAPPING-LEVEL = { 1.0 | 1.1 | 1.2 | 2.0 | 2.1 | 2.2 | 3.0 | 4.0 | 4.1 | 4.2 | 4.3 }
XML バインディングおよび言語構造の生成時に支援機能で使用するマッピング・レベルを指定します。 利用可能な最新のマッピング・レベルを使用することをお勧めします。DFHSC2LS の場合、マッピング・レベル 3.0 以上を使用することをお勧めします。

3.0 xsd:dateTime データ型は CICS ASKTIME フォーマットにマップされます。

4.0 UTF-16 を使用する場合には、CICS TS 5.2 以降の領域でこのマッピング・レベルを使用します。

4.1 切り捨て可能な配列をサポートするには、CICS TS 5.2 以降の領域でこのマッピング・レベルを使用します。

4.2 追加プロパティーについては、CICS TS 5.4 以降の領域でこのマッピング・レベルを使用します。

4.3 多次元配列をサポートするには、CICS TS 5.4 以降の領域でこのマッピング・レベルを使用します。

MAPPING-OVERRIDES = { SAME-AS-MAPPING-LEVEL | HYPHENS-AS-UNDERSCORES | NO-ARRAY-NAME-INDEXING | UNDERSCORES-AS-HYPHENS }
言語構造を生成するときの指定されたマッピング・レベルについて、デフォルトの動作を指定変更するかどうかを指定します。

SAME-AS-MAPPING-LEVEL
このパラメーターは、マッピング・レベルと同じスタイルで言語構造を生成します。これはデフォルトです。

HYPHENS-AS-UNDERSCORES
PL/I のみ。このパラメーターは、生成される PL/I 言語構造を読みやすくするために、XML 文書内のすべてのハイフンを文字 X ではなく下

線に変換します。詳しくは、XML スキーマから PL/I へのマッピングを参照してください。このオプションは、マッピング・レベル 4.2 で自動的に有効になります。

NO-ARRAY-NAME-INDEXING
COBOL および Enterprise PL/I のみ。配列内のフィールド名が、構造の上位の範囲内でのみ固有であることを確認してください。

UNDERSCORES-AS-HYPHENS
COBOL のみ。 XML 文書内のあらゆる下線を文字 X ではなくハイフンに変換します。このオプションを使用すると、生成される COBOL 言語構造が読みやすくなります。フィールド名の競合が発生する場合、そのフィールドが必ず固有になるように番号が付きます。 詳しくは、363 ページの『XML スキーマとCOBOL のマッピング』を参照してください。

このオプションは、マッピング・レベル 4.0 で自動的に有効になります。

MINIMUM-RUNTIME-LEVEL = { MINIMUM | 3.0 | 4.1 | 4.2 |4.3 | CURRENT }
XML バインディングをデプロイすることが可能な、CICS の最小ランタイム環境を指定します。 指定した他のパラメーターと一致しないレベルを選択すると、エラー・メッセージを受け取ります。選択可能なオプションは以下のとおりです。

MINIMUM
選択したパラメーターを前提として、最低限可能な CICS 実行時レベルが自動的に割り振られます。

3.0 CICS XML 支援機能を使用して、高機能のデータ・マッピングを利用する場合は、ランタイム・レベル 3.0 以上を指定します。

4.0 生成された Web サービス・バインディング・ファイルは、CICS TS 5.2 以降の領域に正常にデプロイされます。この実行時レベルでは、**MAPPING-LEVEL** パラメーターにマッピング・レベル 4.0 以前を使用できます。このレベルでは任意のオプション・パラメーターを使用できます。

4.1 生成された Web サービス・バインディング・ファイルは、CICS TS 5.2 以降の領域に正常にデプロイされます。この実行時レベルでは、**MAPPING-LEVEL** パラメーターには、マッピング・レベル 4.1 以前を使用することができます。

4.2 生成された Web サービス・バインディング・ファイルは、CICS TS V5.4 以降の領域に正常にデプロイされます。この実行時レベルでは、**MAPPING-LEVEL** パラメーターには、マッピング・レベル 4.2 以前を使用することができます。

4.3 生成された Web サービス・バインディング・ファイルは、CICS TS 5.4 以降の領域に正常にデプロイされます。この実行時レベルでは、**MAPPING-LEVEL** パラメーターには、マッピング・レベル 4.3 以前を使用することができます。

CURRENT
生成された XML バインディングを、ランタイム環境が XML バイン

ディングの生成に使用された領域と同じである CICS 領域にデプロイする場合、このランタイム・レベルを使用します。

OVERWRITE-OUTPUT = **NO** | **YES**
ファイル・システム上の既存の CICS BUNDLE が上書き可能かどうかを制御します。

NO 既存 BUNDLE は何も置き換えられません。 既存 BUNDLE が検出された場合、DFHLS2SC はエラー・メッセージ DFHPI9689E を発行し強制終了します。

YES どの既存 BUNDLE も置き換えられます。 既存の BUNDLE が検出された場合、DFHLS2SC はメッセージ DFHPI9683W を発行し、ファイルが置き換えられたことを通知します。

PDSCP = *value*
区分データ・セット・メンバーで使用されるコード・ページを指定します。ここで、*value* は CCSID 番号または Java コード・ページ番号です。 このパラメーターを指定しない場合、z/OS UNIX システム・サービス・コード・ページが使用されます。例えば、PDSCP=037 を指定することができます。

PDSLIB = *value*
生成された高水準言語が含まれる区分データ・セットの名前を指定します。

PDSMEM = *value*
高水準言語構造が含まれる区分データ・セット・メンバーの名前を生成するために DFHSC2LS が使用する、1 から 6 文字の文字接頭部を指定します。

DFHSC2LS は、操作ごとに 1 つの区分データ・セット・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。

SCHEMA = *value*
XML スキーマの読み込み元になるファイルの、完全修飾された z/OS UNIX 名。 DFHSC2LS はこのファイルを作成しますが、存在していないディレクトリー構造を作成することはありません。

XML スキーマまたは WSDL 文書のいずれかを、 DFHSC2LS への入力データとして使用できます。 入力データのソースを指定するため、このパラメーターまたは **WSDL** パラメーターのいずれかを指定する必要があります。ただし、両方を同時に指定することはできません。

STRUCTURE ={*PDSMEM_value* | *data*}
C および C++ での、最上位データ構造の名前。デフォルト値は **PDSMEM** パラメーターの値です。

TYPES = *value*
有効にするグローバル・タイプのローカル名のリストを定義します。 *value* が ALL である場合は、すべてのグローバル・タイプが有効であることを示します。 デフォルトでは、グローバル・タイプは有効化されていません。

WIDE-COMP3 = { **FULL** | **NO** | **YES** }
生成される COBOL または PL/I 言語構造でパック 10 進数の可変長の最大サイズを制御します。

FULL COBOL および PL/I の場合。DFHJS2LS は、すべての有効な値を保持

できるサイズでパック 10 進数フィールドを生成します。最大サイズは 31 桁です。これはデフォルトです。

NO COBOL のみ。 DFHJS2LS は、COBOL 言語構造タイプ COMP-3 を生成しているとき、パック 10 進数の可変長を 18 に制限します。 パック 10 進数のサイズが 18 より大きい場合、指定されたタイプは合計 18 桁に制限されていることを示すメッセージ DFHPI9022W が出されます。

YES COBOL のみ。 DFHJS2LS は、COBOL 言語構造タイプ COMP-3 を生成しているとき、最大サイズ 31 をサポートします。

注: NO および YES オプションを指定すると、すべての有効な値を表わすことができないフィールドが生成されます。FULL オプションは、この問題を回避します。ただし、FULL オプションを指定すると、パック 10 進数フィールドで無効な値が表される可能性があります。例えば、スキーマに最大サイズとして最大 5 桁の数字と最大 2 桁の小数点以下の数値が指定されている場合、FULL オプションを指定すると、7 桁の数字が許容されるパック 10 進数フィールドが生成されます。この場合、25000 や 999.99 などの有効な値を収容するだけのスペースがあると同時に、9999.99 などの無効な値を収容するだけのスペースも提供されることになります。FULL オプションを使用する場合は、アプリケーション・データに無効な値が生成されないよう注意してください。

WSDL = *value*

WSDL 文書の完全修飾された z/OS UNIX 名。

XML スキーマまたは WSDL 文書のいずれかを、 DFHSC2LS への入力データとして使用できます。 入力データのソースを指定するため、このパラメーターまたは **SCHEMA** パラメーターのいずれかを指定する必要があります。ただし、両方を同時に指定することはできません。

XMLCP = { **LOCAL** | **UTF-8** | **EBCDIC-CP-US** }

XML バインディングを生成するために使用されるコード・ページを指定します。

LOCAL

この値はデフォルトです。XML がローカル・コード・ページを使用して生成され、XML スキーマでエンコード・タグが生成されないことを指定します。

UTF-8

XML が UTF-8 コード・ページを使用して生成されることを指定します。 XML スキーマでエンコード・タグが生成されます。 このオプションを指定する場合、XML スキーマが異なるプラットフォーム間でコピーされるときにエンコードが正しく維持されるようにする必要があります。

EBCDIC-CP-US

XML が US EBCDIC コード・ページを使用して生成されることを指定します。 XML スキーマでエンコード・タグが生成されます。

XSDBIND = *value*

XML バインディングの完全修飾された z/OS UNIX 名。 DFHSC2LS はこの

ファイルを作成しますが、存在していないディレクトリー構造を作成することはありません。 ファイル拡張子は .xsdbind です。

高水準言語と XML のスキーマ・マッピング

CICS アシスタントを使用して高水準言語構造と XML スキーマまたは WSDL ドキュメントとのマッピングを生成します。さらに、CICS アシスタントは、高水準言語構造から XML スキーマまたは WSDL 文書を (またはその逆を) 生成します。

ユーティリティー・プログラム DFHSC2LS と DFHLS2SC を合わせて CICS XML アシスタントといいます。 ユーティリティー・プログラム DFHWS2LS と DFHLS2WS を合わせて CICS Web サービス・アシスタントといいます。

- DFHLS2SC および DFHLS2WS は高水準言語構造を XML スキーマと WSDL 文書にそれぞれマップします。
- DFHSC2LS および DFHWS2LS は XML スキーマと WSDL 文書を高水準言語構造にマップします。

以下に示すように、2 つのマッピングは対称的ではありません。

- DFHLS2SC または DFHLS2WS を使って言語データ構造を処理した結果として生成される XML スキーマまたは WSDL 文書を、相補的なユーティリティー・プログラム (それぞれ DFHSC2LS または DFHWS2LS) を使って処理する場合、最終的な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。ただし、最終的なデータ構造は、最初のデータ構造と論理的には同等です。
- DFHSC2LS または DFHWS2LS を使って XML スキーマまたは WSDL 文書を処理した結果として生成される言語構造を、相補的なユーティリティー・プログラム (それぞれ DFHLS2SC または DFHLS2WS) を使って処理する場合、最終的な XML スキーマまたは WSDL 文書の中の XML スキーマが最初のものと同じになることは期待できません。
- 場合によっては、DFHLS2SC および DFHLS2WS でサポートされない言語構造が DFHSC2LS および DFHWS2LS によって生成されることがあります。

DFHLS2SC および DFHLS2WS で処理される言語構造は、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。

CICS アシスタントのデータ・マッピングの制限**:**

CICS は高水準言語構造と XML スキーマ、または高水準言語と WSDL バージョン 1.1 もしくは 2.0 に準拠する WSDL 文書との間の双方向のデータ・マッピングを制限付きでサポートしています。 これらの制限は DFHWS2LS および DFHSC2LS ツールにのみ適用され、マッピング・レベルによって異なります。

全マッピング・レベルでの制限

- 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。 そのため、use 属性を literal の値に設定する必要があります。 use="encoded" はサポートされません。

- 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。 スキーマ内では、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。
- 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。
- Web サービス記述で定義された SOAP 障害はすべて無視されます。サービス・プロバイダー・アプリケーションで SOAP 障害メッセージを送信するには、**EXEC CICS SOAPFAULT** コマンドを使用します。
- DFHWS2LS および DFHSC2LS が、特定のスコープでサポートする <xsd:any> エレメントは 1 つまでです。 例えば、次のスキーマ・フラグメントはサポートされません。

  ```
  <xsd:sequence>
  <xsd:any/>
  <xsd:any/>
  </xsd:sequence>
  ```

  ここで、必要に応じて <xsd:any> を使用して minOccurs および maxOccurs を指定できます。 例えば、次のスキーマ・フラグメントはサポートされます。

  ```
  <xsd:sequence>
  <xsd:any minOccurs="2" maxOccurs="2"/>
  </xsd:sequence>
  ```
- 循環参照はサポートされません。 例えば、型 A に型 B が含まれ、さらには型 B に型 A が含まれる場合などです。
- グループ・エレメント (<xsd:choice>、<xsd:sequence>、<xsd:group>、または <xsd:all> エレメント) などにおける繰り返しはサポートされません。 例えば、次のスキーマ・フラグメントはサポートされません。

  ```
  <xsd:choice maxOccurs="2">
  <xsd:element name="name1" type="string"/>
  </xsd:choice>
  ```

  例外として、マッピング・レベル 2.1 以上では、これらのエレメントに対して maxOccurs="1" および minOccurs="0" がサポートされます。
- DFHSC2LS および DFHWS2LS は、xsi:type 属性または置換グループの XML スキーマで宣言されているデータ・タイプおよびエレメントから派生した、SOAP メッセージ内のデータ・タイプおよびエレメントをサポートしません。マッピング・レベルが 2.2 以上で、親エレメントまたは親タイプが抽象であると定義されている場合を除きます。
- <xsd:choice> エレメントに組み込まれた <xsd:sequence> エレメントおよび <xsd:group> エレメントは、マッピング・レベル 2.2 より前のレベルではサポートされません。 <xsd:choice> エレメントに組み込まれた <xsd:choice> エレメントおよび <xsd:all> エレメントは一切サポートされません。

マッピング・レベル **1.1** 以上におけるサポートの向上

マッピング・レベルが 1.1 以上である場合、DFHWS2LS は次の XML エレメントおよびエレメント属性をサポートします。

- <xsd:list> エレメント。
- <xsd:union> エレメント。

- xsd:anySimpleType タイプ。
- <xsd:attribute> エレメント。 マッピング・レベル 1.0 ではこのエレメントは無視されます。

マッピング・レベル **2.1** 以上におけるサポートの向上

マッピング・レベルが 2.1 以上である場合、DFHWS2LS は次の XML エレメントおよびエレメント・タイプをサポートします。

- <xsd:any> エレメント。
- xsd:anyType タイプ。
- 抽象エレメント。 それより前のマッピング・レベルでは、抽象エレメントは継承の階層の非端末型としてのみサポートされていました。
- <xsd:all>、<xsd:choice>、および <xsd:sequence> エレメントにおける maxOccurs および minOccurs 属性。maxOccurs="1" および minOccurs="0" の場合のみ。
- COBOL における "FILLER" フィールドおよび PL/I における "*" フィールドは抑制されます。フィールドは生成された WSDL に表示されず、相応のギャップが実行時にデータ構造に残されます。

マッピング・レベル **2.2** 以上におけるサポートの向上

マッピング・レベルが 2.2 以上の場合、DFHSC2LS と DFHWS2LS による <xsd:choice> エレメントのサポートが向上し、<xsd:choice> エレメントで最大 255 のオプションがサポートされます。 <xsd:choice> サポートについて詳しくは、399 ページの『<xsd:choice> のサポート』を参照してください。

マッピング・レベル 2.2 以上では、CICS アシスタントが次の XML マッピングをサポートします。

- 置換グループ
- エレメントの固定値
- 抽象データ・タイプ

<xsd:choice> エレメントに組み込まれた <xsd:sequence> エレメントおよび <xsd:group> エレメントはマッピング・レベル 2.2 以上でサポートされています。例えば、次のスキーマ・フラグメントはサポートされます。

```
<xsd:choice>
<xsd:element name="name1" type="string"/>
<xsd:sequence/>
</xsd:choice>
```

SOAP メッセージの親エレメントまたは親タイプが抽象と定義されている場合、DFHSC2LS および DFHWS2LS は XML スキーマの宣言されたデータ・タイプおよびエレメントから派生したデータ・タイプおよびエレメントをサポートします。

マッピング・レベル **3.0** 以上におけるサポートの向上

マッピング・レベルが 3.0 以上の場合、CICS アシスタントが次のマッピングの向上をサポートします。

- DFHSC2LS および DFHWS2LS は、xsd:dateTime データ型を CICS ASKTIME 形式にマップします。
- DFHLS2WS は、WSDL 文書および Web サービス・バインディングを、1 つのコンテナーのみでなく、多数のコンテナーを使用するアプリケーションから生成できます。
- 固定長データ構造で記述される切り捨てられたデータの許容。この動作は、CICS 支援機能で **DATA-TRUNCATION** パラメーターを使用することによって設定できます。

マッピング・レベル **4.0** 以上で強化されたサポート

マッピング・レベルが 4.0 以上の場合、CICS 支援機能により以下のマッピングの改良がサポートされています。

マッピング・レベル 4.0 以上では、DFHLS2SC および DFHLS2WS は、COBOL OCCURS DEPENDING ON 節をサポートし、COBOL 文字配列の XML ストリングへのマッピングをサポートします。この動作を設定するには、CICS 支援機能で、**CHAR-OCCURS** パラメーターを使用します。

- パラメーター **DATA-TRUNCATION=ENABLED** を指定する必要があります。
- 複合 OCCURS DEPENDING ON はサポートされません。この制約は、構造体の最後のフィールドに対してのみ OCCURS DEPENDING ON がサポートされることを意味します。
- CICS は、OCCURS DEPENDING ON 節のターゲットとして修飾名 ('OF' キーワードの使用) をサポートしません (例えば、FIELD1 OF STRUCTURE1)。
- CICS は UNBOUNDED キーワードをサポートしません。アプリケーションによって予期される表の最大サイズを指定する必要があります。

マッピング・レベル 4.0 以上では、CICS Web サービスは、UTF-16 Unicode を使用してエンコードされるアプリケーション・データの変換をサポートします。

- DFHLS2WS または DFHLS2SC を使用する場合、UTF-16 に言語固有のデータ型を使用することによって、この動作を使用可能にできます。
- DFHWS2LS または DFHSC2LS を使用する場合、CCSID=1200 を設定することによって、この動作を使用可能にできます。
- CICS は、単一の Unicode コード・ページ、即ち IBM プライベート使用領域を使用する UTF-16BE (CCSID 1200) のみをサポートします。
- UTF-8 を使用してエンコードされたアプリケーション・データの変換はサポートされていません。

注: DFHLS2WS および DFHLS2SC は、COBOL の GROUP USAGE NATIONAL 節をサポートしていません。

COBOL から **XML** スキーマへのマッピング:

DFHLS2SC および DFHLS2WS ユーティリティー・プログラムは COBOL データ構造と XML スキーマ定義との間のマッピングをサポートします。

以下の規則にしたがって、COBOL 名が XML 名に変換されます。

1. 重複する名前は、1 つ以上の数字が追加されて固有の名前になります。

例えば、year の 2 つのインスタンスは year と year1 になります。

2. ハイフンは、下線文字に置き換えられます。 一連の連続するハイフンは、連続する下線で置き換えられます。

   例えば、current-user--id は current_user__id になります。

3. ハイフンで区切られており、大文字のみを含む名前のセグメントは、小文字に変換されます。

   例えば、CA-REQUEST-ID は ca_request_id になります。

4. 数字で始まる名前には、先頭に下線文字が追加されます。

   例えば、9A-REQUEST-ID は _9a_request_id になります。

CICS 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。 表に示されていない COBOL データ記述エレメントは、DFHLS2SC または 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 および 77 のデータ記述項目はサポートされていません。 レベル番号が 88 のデータ記述項目は無視されます。
- データ記述項目における以下の節は、サポートされません。

  REDEFINES

  RENAMES (レベル 66)

  DATE FORMAT
- データ記述項目における以下の節は、無視されます。

  BLANK WHEN ZERO

  JUSTIFIED

  VALUE
- SIGN 文節 SIGN TRAILING はサポートされます。 SIGN 文節 SIGN LEADING は、DFHLS2SC または DFHLS2WS で指定されたマッピング・レベルが 1.2 以上の場合のみサポートされます。
- SIGN TRAILING 文節と SIGN LEADING 文節の両方では、SEPARATE CHARACTER はマッピング・レベル 1.2 以上でサポートされます。
- USAGE 節における以下の句は、サポートされません。

  OBJECT REFERENCE

  POINTER

  FUNCTION-POINTER

  PROCEDURE-POINTER
- USAGE 文節の次の句は、マッピング・レベル 1.2 以上でサポートされます。

  COMPUTATIONAL-1

  COMPUTATIONAL-2
- 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、S、および Z です。
- PACKED-DECIMAL データ記述項目でサポートされる PICTURE 文字は、9、S、V、および Z です。

- 編集済みの数値データ記述項目でサポートされている PICTURE 文字は、9 と Z だけです。
- **MAPPING-LEVEL** パラメーターを 1.2 以上に設定して、**CHAR-VARYING** パラメーターを NULL に設定すると、文字配列は xsd:string にマップされ、ヌル終了ストリングとして処理されます。
- **MAPPING-LEVEL** パラメーターを 1.2 以上に設定して、**CHAR-VARYING** パラメーターを BINARY に設定すると、文字配列は xsd:base64Binary にマップされ、バイナリー・データとして処理されます。
- **MAPPING-LEVEL** パラメーターを 1.2 以上に設定して、**CHAR-VARYING** パラメーターを COLLAPSE に設定すると、ストリングの末尾の空白文字は無視されます。
- OCCURS DEPENDING ON 節は、マッピング・レベル 4.0 以上でサポートされます。複雑な OCCURS DEPENDING ON はサポートされません。つまり、構造の最後のフィールドにのみ OCCURS DEPENDING ON がサポートされます。
- OCCURS INDEXED BY 節は、すべてのマッピング・レベルでサポートされます。
- OCCURS 句は最大 65535 TIMES のオカレンスをサポートします。つまり、OCCURS *n* TIMES (ここで、*n* は 65535 より大きい値) はサポートされません。

COBOL のデータ記述	スキーマの **simpleType**
PIC X(*n*) PIC A(*n*) PIC G(*n*) DISPLAY-1 PIC N(*n*)	<xsd:simpleType> <xsd:restriction base="xsd:string"> <xsd:maxlength value=" *n*"/> <xsd:whiteSpace value="preserve"/> </xsd:restriction> </xsd:simpleType>
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 ここで、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 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。	<xsd:simpleType> <xsd:restriction base="xsd:unsignedShort"> <xsd:minInclusive value="0"/> <xsd:maxInclusive value=" *n*"/> </xsd:restriction> </xsd:simpleType>
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 ここで、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 PIC S9(*n*) COMP-4 PIC S9(*n*) COMP-5 PIC S9(*n*) BINARY ここで、*n* ≤ 4.	<xsd:simpleType> <xsd:restriction base="xsd:short"> </xsd:restriction> </xsd:simpleType>
PIC S9(*n*) COMP PIC S9(*n*) COMP-4 PIC S9(*n*) COMP-5 PIC S9(*n*) BINARY ここで、 5 ≤ *n* ≤ 9.	<xsd:simpleType> <xsd:restriction base="xsd:int"> </xsd:restriction> </xsd:simpleType>

COBOL のデータ記述	スキーマの **simpleType**
PIC S9(*n*) COMP PIC S9(*n*) COMP-4 PIC S9(*n*) COMP-5 PIC S9(*n*) BINARY ここで、9 < *n*	<xsd:simpleType> <xsd:restriction base="xsd:long"> </xsd:restriction> </xsd:simpleType>
PIC 9(*n*) COMP PIC 9(*n*) COMP-4 PIC 9(*n*) COMP-5 PIC 9(*n*) BINARY ここで、*n* ≤ 4.	<xsd:simpleType> <xsd:restriction base="xsd:unsignedShort"> </xsd:restriction> </xsd:simpleType>
PIC 9(*n*) COMP PIC 9(*n*) COMP-4 PIC 9(*n*) COMP-5 PIC 9(*n*) BINARY ここで、 5 ≤ *n* ≤ 9.	<xsd:simpleType> <xsd:restriction base="xsd:unsignedInt"> </xsd:restriction> </xsd:simpleType>
PIC 9(*n*) COMP PIC 9(*n*) COMP-4 PIC 9(*n*) COMP-5 PIC 9(*n*) BINARY ここで、9 < *n*	<xsd:simpleType> <xsd:restriction base="xsd:unsignedLong"> </xsd:restriction> </xsd:simpleType>

COBOL のデータ記述	スキーマの **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*
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*
PIC S9(*m*) COMP-3 DATETIME=PACKED15 の場合マッピング・レベル 3.0 でサポートされる	<xsd:simpleType> <xsd:restriction base="xsd:dateTime" </xsd:restriction> </xsd:simpleType> タイム・スタンプの形式は CICS ABSTIME です。
PIC S9(*m*)V9(*n*) DISPLAY マッピング・レベル 1.2 以上でサポートされる	<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*
COMP-1 マッピング・レベル 1.2 以上でサポートされる **注:** IBM Hexadecimal Floating Point (HFP) データ表記は、XML に使用される IEEE-754-1985 表記とまったく同じではありません。一部の値が 1 つ表記からもう 1 つの表記に正確に変換されない可能性があります。極端に大きいまたは小さい値は、xsd:float データ型に対して有効ではない場合があります。 HFP 表記との間の変換時に、一部の値で精度が失われる可能性があります。正確な変換が重要になる場合、COMP-1 データ型を使用せずに、固定精度のデータ型を代わりに使用することを検討してください。	<xsd:simpleType> <xsd:restriction base="xsd:float"> </xsd:restriction> </xsd:simpletype>
COMP-2 マッピング・レベル 1.2 以上でサポートされる **注:** IBM Hexadecimal Floating Point (HFP) データ表記は、XML に使用される IEEE-754-1985 表記とまったく同じではありません。一部の値が 1 つ表記からもう 1 つの表記に正確に変換されない可能性があります。極端に大きいまたは小さい値は、xsd:double データ型に対して有効ではない場合があります。 HFP 表記との間の変換時に、一部の値で精度が失われる可能性があります。正確な変換が重要になる場合、COMP-2 データ型を使用せずに、固定精度のデータ型を代わりに使用することを検討してください。	<xsd:simpleType> <xsd:restriction base="xsd:double"> </xsd:restriction> </xsd:simpletype>
data description OCCURS *n* TIMES	<xsd:element name= *"field-name"* minOccurs= *"n"* maxOccurs= *"n"*> ... </xsd:element> エレメントの内容は、使用されるデータ型によって異なります。
data description OCCURS *n* TO *m* TIMES DEPENDING ON *t* マッピング・レベル 4.0 でサポート	<xsd:element name= *"field-name"* minOccurs= *"n"* maxOccurs= *"m"*> ... </xsd:element>

COBOL のデータ記述	スキーマの **simpleType**
PIC X OCCURS *n* TIMES PIC A OCCURS *n* TIMES PIC G DISPLAY-1 OCCURS *n* TIMES PIC N OCCURS *n* TIMES	**CHAR-OCCURS** =STRING の場合: <xsd:element name= *"field-name"* > <xsd:simpleType> <xsd:restriction base= *"s:string"*> <xsd:maxLength value= *"n"*> </xsd:restriction> </xsd:simpleType> </xsd:element> これはストリングです。 **CHAR-OCCURS** =ARRAY の場合: <xsd:element name= *"field-name"* minOccurs= *"n"* maxOccurs= *"n"*> <xsd:simpleType> <xsd:restriction base= *"s:string"*> <xsd:maxLength value= *"1"* </xsd:restriction> </xsd:simpleType> </xsd:element> これは単一文字の配列です。
PIC X OCCURS *n* TO *m* TIMES DEPENDING ON *t* PIC A OCCURS *n* TO *m* TIMES DEPENDING ON *t* PIC G DISPLAY-1 OCCURS *n* TO *m* TIMES DEPENDING ON *t* PIC N OCCURS *n* TO *m* TIMES DEPENDING ON *t*	**CHAR-OCCURS** =STRING の場合: <xsd:element name= *"field-name"* > <xsd:simpleType> <xsd:restriction base= *"s:string"*> <xsd:maxLength value= *"m"*> <xsd:minLength value= *"n"*> </xsd:restriction> </xsd:simpleType> </xsd:element>
PIC N(*n*) USAGE NATIONAL **CHAR-USAGE** =NATIONAL の場合: PIC N(*n*)	<xsd:simpleType> <xsd:restriction base="xsd:string"> <xsd:maxlength value="n"/> <xsd:whiteSpace value="preserve"/> </xsd:restriction> </xsd:simpleType> 実行時に、CICS はアプリケーション・データ構造フィールドに UTF-16 データを取り込みます。

XML スキーマと**COBOL** のマッピング**:**

DFHSC2LS および DFHWS2LS のユーティリティー・プログラムは、XML スキーマ定義および COBOL データ構造との間のマッピングをサポートします。

CICS 支援機能は以下の規則を使用して、スキーマのエレメント名から COBOL 変数の固有かつ有効な名前を生成します。

1. COBOL 予約語には接頭部「X」が付きます。

   例えば、DISPLAY は XDISPLAY になります。

2. A から Z、a から z、0 から 9、またはハイフン以外の文字は、「X」で置き換えられます。

   例えば、 monthly_total は monthlyXtotal となります。 **MAPPING-OVERRIDES** パラメーターを使用して、他の文字の処理方法を変更できます。 例えば、値 UNDERSCORES-AS-HYPHENS を設定した場合、XML の下線はすべて X ではなくハイフンに変換されます。したがって、monthly_total は monthly-total になります。

3. 最後の文字がハイフンである場合は、「X」で置き換えられます。

   例えば、ca-request- は ca-requestX になります。

4. 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:element で minOccurs および maxOccurs を異なる値で指定する場合)、スキーマ・エレメント名が 23 文字を超えると、この長さに切り捨てられます。

   スキーマで変数が固定の基数を持つように指定する場合、スキーマ・エレメント名が 28 文字を超えると、この長さに切り捨てられます。

5. 同じスコープ内の重複した名前は、名前の 2 番目以降のインスタンスに 1 つまたは 2 つの数字を追加することによって固有にします。

   例えば、 year の 3 つのインスタンスは year 、 year1 、および year2 になります。

   もし上記の動作が望ましくないなら、ユーザーはユーティリティーの入力として MAPPING-OVERRIDES=NO-ARRAY-NAME-INDEXING を指定できます。これにより、名前の 2 番目以降のインスタンスへの 1 つまたは 2 つの数字の追加が無効になります。

6. 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 (minOccurs および maxOccurs を異なる値で指定する場合) 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。

   詳しくは、 389 ページの『変化するエレメントの配列』を参照してください。

7. 属性では、前の規則がエレメント名に適用されます。 接頭部 attr- がエレメント名に追加され、この後に -value または -exist が付きます。全長が 28 文字を超える場合、エレメント名は切り捨てられます。 詳しくは、 394 ページの『XML 属性のサポート』を参照してください。

   nillable 属性には特別な規則があります。接頭部 attr- が追加されますが、エレメント名の先頭には nil- も追加されます。 エレメント名の後には -value が続きます。 全長が 28 文字を超える場合、エレメント名は切り捨てられます。

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。

DFHSC2LS および DFHWS2LS は、指定されたマッピング・レベルを使用して、スキーマ・タイプを次の表に従って、 COBOL のデータ記述エレメントにマップします。次の点に注意してください。

- **MAPPING-LEVEL** パラメーターを 1.2 以上に設定して、**CHAR-VARYING** パラメーターを NULL に設定すると、可変長文字データはヌル終了ストリングにマップされ、ヌル終止符として追加された 1 つの文字が割り振られます。
- **MAPPING-LEVEL** パラメーターを 1.2 以上に設定して、**CHAR-VARYING** パラメーターを YES に設定すると、可変長文字データは 2 つの関連エレメント (長さフィールドとデータ・フィールド) にマップされます。以下に例を示します。

```
<xsd:simpleType name="VariableStringType">
<xsd:restriction base="xsd:string">
<xsd:minLength value="1"/>
<xsd:maxLength value="10000"/>
</xsd:restriction>
</xsd:simpleType>
<xsd:element name="textString" type="tns:VariableStringType"/>
```

これは、次にマップします。

```
15 textString-length PIC S9999 COMP-5 SYNC
15 textString PIC X(10000)
```

スキーマの単純型	**COBOL** のデータ記述

`<xsd:simpleType>`
`<xsd:restriction base="xsd:anyType">`
`</xsd:restriction>`
`</xsd:simpleType>`
	マッピング・レベル 2.0 以下の場合
	サポートされていない
	マッピング・レベル 2.1 の場合
	サポート対象

`<xsd:simpleType>`
`<xsd:restriction base="xsd:anySimpletype">`
`</xsd:restriction>`
`</xsd:simpleType>`
	マッピング・レベル 1.0 の場合
	サポートされていない
	マッピング・レベル 1.1 以上の場合
	`PIC X(255)`

`<xsd:simpleType>`
`<xsd:restriction base="xsd:`
type"
`<xsd: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
	すべてのマッピング・レベル:
	`PIC X(`
	z
	`)`

スキーマの単純型	**COBOL** のデータ記述
<xsd:simpleType> <xsd:restriction base="xsd: *type*"> <xsd: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	マッピング・レベル 4.0 以上で **CCSID=1200** の場合: PIC N(*z*) USAGE NATIONAL
<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 time gDay gMonth gYear gMonthDay gYearMonth	すべてのマッピング・レベル: PIC X(32)
<xsd:simpleType> <xsd:restriction base="xsd:dateTime" </xsd:restriction> </xsd:simpleType>	マッピング・レベル 1.2 以下の場合 PIC X(32) マッピング・レベル 2.0 以上の場合 PIC X(40) マッピング・レベル 3.0 以上の場合 PIC S9(15) COMP-3 形式は CICS ABSTIME です。
<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	すべてのマッピング・レベル: PIC X DISPLAY
<xsd:simpleType> <xsd:restriction base="xsd:short"> </xsd:restriction> </xsd:simpleType>	すべてのマッピング・レベル: PIC S9999 COMP-5 SYNC または PIC S9999 DISPLAY
<xsd:simpleType> <xsd:restriction base="xsd:unsignedShort"> </xsd:restriction> </xsd:simpleType>	すべてのマッピング・レベル: PIC 9999 COMP-5 SYNC または PIC 9999 DISPLAY

スキーマの単純型	**COBOL** のデータ記述
<xsd:simpleType> <xsd:restriction base="xsd:integer"> </xsd:restriction> </xsd:simpleType>	すべてのマッピング・レベル: PIC S9(18) COMP-3
<xsd:simpleType> <xsd:restriction base="xsd:int"> </xsd:restriction> </xsd:simpleType>	すべてのマッピング・レベル: PIC S9(9) COMP-5 SYNC または PIC S9(9) DISPLAY
<xsd:simpleType> <xsd:restriction base="xsd:unsignedInt"> </xsd:restriction> </xsd:simpleType>	すべてのマッピング・レベル: PIC 9(9) COMP-5 SYNC または PIC 9(9) DISPLAY
<xsd:simpleType> <xsd:restriction base="xsd:long"> </xsd:restriction> </xsd:simpleType>	すべてのマッピング・レベル: PIC S9(18) COMP-5 SYNC または PIC S9(18) DISPLAY
<xsd:simpleType> <xsd:restriction base="xsd:unsignedLong"> </xsd:restriction> </xsd:simpleType>	すべてのマッピング・レベル: 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>	すべてのマッピング・レベル: WIDE-COMP3=FULL の場合: PIC 9(*m*)V9(*n*) COMP-3 それ以外の場合: PIC 9(*p*)V9(*n*) COMP-3 ここで、*p* = *m* - *n* です。
<xsd:simpleType> <xsd:restriction base="xsd:boolean"> </xsd:restriction> </xsd:simpleType>	すべてのマッピング・レベル: PIC X DISPLAY 値 x'00' は false を示し、x'01' は true を示します。
<xsd:simpleType> <xsd:list> <xsd:simpleType> <xsd:restriction base="xsd:int"/> </xsd:simpleType> </xsd:list> </xsd:simpleType>	マッピング・レベル 1.0 の場合 サポートされていない マッピング・レベル 1.1 以上の場合 PIC X(255)

スキーマの単純型	**COBOL** のデータ記述
<xsd:simpleType> <xsd:union memberTypes="xsd:int xsd:string"/> </xsd:simpleType>	マッピング・レベル 1.0 の場合 サポートされていない マッピング・レベル 1.1 以上の場合 `PIC X(255)`

<xsd:simpleType>
<xsd:restriction base="xsd:base64Binary">
<xsd:length value="
z"/>
</xsd:restriction>
</xsd:simpleType>
 	
<xsd:simpleType>
<xsd:restriction base="xsd:base64Binary">
</xsd:restriction>
</xsd:simpleType>
 	
長さは定義されていません。
	マッピング・レベル 1.0 の場合
	
	サポートされていない
	
	マッピング・レベル 1.1 の場合
	
	`PIC X(`
	y
	`)`
	
	ここで、*y* =4×(ceil(z/3)) です。ceil(*x*) は *x* 以上の最小の整数です。
	
	マッピング・レベル 1.2 以上の場合
	
	`PIC X(`
	z
	`)`
	
	この場合、長さは固定されています。
	
	`PIC X(16)`
	この場合、長さは定義されていません。 このフィールドは、2 進データが保管されるコンテナーの 16 バイトの名前を保持します。

<xsd:simpleType>
<xsd:restriction base="xsd:float">
</xsd:restriction>
</xsd:simpletype>
	マッピング・レベル 1.1 以下の場合
	
	`PIC X(32)`
	
	マッピング・レベル 1.2 以上の場合
	
	`COMP-1`
	
	注: IBM Hexadecimal Floating Point (HFP) データ表記は、XML に使用される IEEE-754-1985 表記とまったく同じではありません。一部の値が 1 つ表記からもう 1 つの表記に正確に変換されない可能性があります。極端に大きいまたは小さい値は、`xsd:float` データ型に対して有効ではない場合があります。 HFP 表記との間の変換時に、一部の値で精度が失われる可能性があります。正確な変換が重要になる場合、COMP-1 データ型を使用せずに、固定精度のデータ型を代わりに使用することを検討してください。

<xsd:simpleType>
<xsd:restriction base="xsd:double">
</xsd:restriction>
</xsd:simpletype>
	マッピング・レベル 1.1 以下の場合
	
	`PIC X(32)`
	
	マッピング・レベル 1.2 以上の場合
	
	`COMP-2`
	
	注: IBM Hexadecimal Floating Point (HFP) データ表記は、XML に使用される IEEE-754-1985 表記とまったく同じではありません。一部の値が 1 つ表記からもう 1 つの表記に正確に変換されない可能性があります。極端に大きいまたは小さい値は、`xsd:double` データ型に対して有効ではない場合があります。 HFP 表記との間の変換時に、一部の値で精度が失われる可能性があります。正確な変換が重要になる場合、COMP-2 データ型を使用せずに、固定精度のデータ型を代わりに使用することを検討してください。

表に示したスキーマ・タイプの一部は、`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* は数字 9 からなるストリングです。

- 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* は数字 9 からなるストリングです。

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++** から **XML** スキーマへのマッピング**:**

DFHLS2SC および DFHLS2WS のユーティリティー・プログラムは C および C++ データ・タイプと XML スキーマ定義との間のマッピングをサポートします。

C および C++ の名前は、次の規則に従って XML の名前に変換されます。

1. XML エレメント名で無効な文字は、「X」で置き換えられます。

   例えば、 monthly-total は monthlyXtotal となります。

2. 重複する名前は、1 つ以上の数字が追加されて固有の名前になります。

   例えば、year の 2 つのインスタンスは year と year1 になります。

DFHLS2SC および 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++ のタイプは DFHLS2SC または DFHLS2WS ではサポートされません。 _Packed 修飾子が構造用にサポートされています。 以下の制限が適用されます。

- ヘッダー・ファイルには、最上位の struct インスタンスを含める必要があります。
- 自身をメンバーとして含む構造化タイプを宣言することはできません。
- 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
  long double
  wchar_t (C++ のみ)
  ```

- 以下の文字は、ヘッダー・ファイルに含まれている場合、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)

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、以下の項目を指定できません。

  共用体

  クラス宣言

  列挙型データ・タイプ

  ポインター型変数

  テンプレート宣言

  定義済みマクロ (名前の先頭と末尾が 2 つの下線文字 (__) 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、32 ビットのコンパイラーを使用して int が 4 バイトに確実にマップされるようにする必要があります。
- 次の C++ 予約済みキーワードはサポートされません。

  explicit

  using

  namespace

  typename

  typeid
- **MAPPING-LEVEL** パラメーターを 1.2 以上に設定して、**CHAR-VARYING** パラメーターを NULL に設定すると、文字配列は xsd:string にマップされ、ヌル終了ストリングとして処理されます。
- **MAPPING-LEVEL** パラメーターを 1.2 以上に設定して、**CHAR-VARYING** パラメーターを BINARY に設定すると、文字配列は xsd:base64Binary にマップされ、バイナリー・データとして処理されます。
- **MAPPING-LEVEL** パラメーターを 1.2 以上に設定して、**CHAR-VARYING** パラメーターを COLLAPSE に設定すると、<xsd:whiteSpace value="collapse"/> がストリング用に生成されます。

C および **C++** のデータ・タイプ	スキーマの **simpleType**
char[*z*]	<xsd:simpleType> <xsd:restriction base="xsd:string"> <xsd:length value=" *z*"/> </xsd:restriction> </xsd:simpletype>
char16_t[*n*]	マッピング・レベル 4.0 以上の場合: <xsd:simpleType> <xsd:restriction base="xsd:string"> <xsd:maxlength value="n"/> <xsd:whiteSpace value="preserve"/> </xsd:restriction> </xsd:simpleType> 実行時に、CICS はアプリケーション・データ構造フィールドに UTF-16 データを取り込みます。
char[8] DATETIME=PACKED15 の場合マッピング・レベル 3.0 以上でサポートされる	<xsd:simpleType> <xsd:restriction base="xsd:dateTime" </xsd:restriction> </xsd:simpleType> タイム・スタンプのフォーマットは CICS ABSTIME です。
char	<xsd:simpleType> <xsd:restriction base="xsd:byte"> </xsd:restriction> </xsd:simpletype>
unsigned char	<xsd:simpleType> <xsd:restriction base="xsd:unsignedByte"> </xsd:restriction> </xsd:simpletype>
short	<xsd:simpleType> <xsd:restriction base="xsd:short"> </xsd:restriction> </xsd:simpletype>
unsigned short	<xsd:simpleType> <xsd:restriction base="xsd:unsignedShort"> </xsd:restriction> </xsd:simpletype>
int long	<xsd:simpleType> <xsd:restriction base="xsd:int"> </xsd:restriction> </xsd:simpletype>
unsigned int unsigned long	<xsd:simpleType> <xsd:restriction base="xsd:unsignedInt"> </xsd:restriction> </xsd:simpletype>
long long	<xsd:simpleType> <xsd:restriction base="xsd:long"> </xsd:restriction> </xsd:simpletype>
unsigned long long	<xsd:simpleType> <xsd:restriction base="xsd:unsignedLong"> </xsd:restriction> </xsd:simpletype>

C および **C++** のデータ・タイプ	スキーマの **simpleType**

`bool`
 	
(C++ のみ)
	`<xsd:simpleType>`
	`<xsd:restriction base="xsd:boolean">`
	`</xsd:restriction>`
	`</xsd:simpletype>`

`float`
 	
マッピング・レベル 1.2 以上でサポートされる
	`<xsd:simpleType>`
	`<xsd:restriction base="xsd:float">`
	`</xsd:restriction>`
	`</xsd:simpletype>`
	
	注: IBM Hexadecimal Floating Point (HFP) データ表記は、XML に使用される IEEE-754-1985 表記とまったく同じではありません。一部の値が 1 つ表記からもう 1 つの表記に正確に変換されない可能性があります。極端に大きいまたは小さい値は、`xsd:float` データ型に対して有効ではない場合があります。 HFP 表記との間の変換時に、一部の値で精度が失われる可能性があります。正確な変換が重要になる場合、float データ型を使用せずに、固定精度のデータ型を代わりに使用することを検討してください。

`double`
 	
マッピング・レベル 1.2 以上でサポートされる
	`<xsd:simpleType>`
	`<xsd:restriction base="xsd:double">`
	`</xsd:restriction>`
	`</xsd:simpletype>`
	
	注: IBM Hexadecimal Floating Point (HFP) データ表記は、XML に使用される IEEE-754-1985 表記とまったく同じではありません。一部の値が 1 つ表記からもう 1 つの表記に正確に変換されない可能性があります。極端に大きいまたは小さい値は、`xsd:double` データ型に対して有効ではない場合があります。 HFP 表記との間の変換時に、一部の値で精度が失われる可能性があります。正確な変換が重要になる場合、double データ型を使用せずに、固定精度のデータ型を代わりに使用することを検討してください。

XML スキーマから **C** および **C++** へのマッピング**:**

DFHSC2LS および DFHWS2LS のユーティリティー・プログラムは各 Web サービス記述に組み込まれている XML スキーマ定義と、C および C++ データ・タイプとの間のマッピングをサポートします。

CICS 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++ 変数の固有で有効な名前を生成します。

1. A から Z、a から z、0 から 9、または _ 以外の文字は、「`X`」で置き換えられます。

   例えば、 `monthly-total` は `monthlyXtotal` となります。
2. 先頭文字が英字ではない場合は、先頭文字が「`X`」で置き換えられます。

   例えば、`_monthlysummary` は `Xmonthlysummary` になります。
3. 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。
4. 同じスコープ内の重複した名前は、1 つ以上の数字を追加することによって固有にします。

   例えば、`year` の 2 つのインスタンスは `year` と `year1` になります。

5. 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:element で minOccurs および 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。

   詳しくは、 389 ページの『変化するエレメントの配列』を参照してください。

6. 属性では、前の規則がエレメント名に適用されます。 接頭部 attr_ がエレメント名に追加され、この後に _value または _exist が付きます。全長が 28 文字を超える場合、エレメント名は切り捨てられます。

   nillable 属性には特別な規則があります。接頭部 attr_ が追加されますが、エレメント名の先頭には nil_ も追加されます。エレメント名の後には _value が付きます。全長が 28 文字を超える場合、エレメント名は切り捨てられます。

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。

DFHSC2LS および DFHWS2LS は、次の表に従って、スキーマ・タイプを C および C++ のデータ・タイプにマップします。 次の規則も適用されます。

- **MAPPING-LEVEL** パラメーターを 1.2 以上に設定して、**CHAR-VARYING** パラメーターを NULL に設定すると、可変長文字データはヌル終了ストリングにマップされ、ヌル終止符として追加された 1 つの文字が割り振られます。
- **MAPPING-LEVEL** パラメーターを 1.2 以上に設定して、**CHAR-VARYING** パラメーターを YES に設定すると、可変長文字データは 2 つの関連エレメント (長さフィールドとデータ・フィールド) にマップされます。

スキーマの **simpleType**	**C** および **C++** のデータ・タイプ
<xsd:simpleType> <xsd:restriction base="xsd:anyType"> </xsd:restriction> </xsd:simpleType>	マッピング・レベル 2.0 以下の場合 サポートされていない マッピング・レベル 2.1 以上の場合 サポート対象
<xsd:simpleType> <xsd:restriction base="xsd:anySimpletype"> </xsd:restriction> </xsd:simpleType>	マッピング・レベル 1.0 の場合 サポートされていない マッピング・レベル 1.1 以上の場合 char[255]

スキーマの **simpleType**	**C** および **C++** のデータ・タイプ
<xsd:simpleType> <xsd:restriction base="xsd: *type*"> <xsd: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	すべてのマッピング・レベル: char[*z*]
<xsd:simpleType> <xsd:restriction base="xsd: *type*"> <xsd: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	マッピング・レベル 4.0 以上で **CCSID=1200** の場合: char16_t[*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 decimal time gDay gMonth gYear gMonthDay gYearMonth	すべてのマッピング・レベル: char[32]
<xsd:simpleType> <xsd:restriction base="xsd:dateTime"> </xsd:restriction> </xsd:simpleType>	マッピング・レベル 1.2 以下の場合 char[32] マッピング・レベル 2.0 以上の場合 char[40] マッピング・レベル 3.0 以上の場合 char[8] タイム・スタンプのフォーマットは CICS ABSTIME です。

スキーマの **simpleType**	**C** および **C++** のデータ・タイプ
<xsd:simpleType> <xsd:restriction base="xsd:byte"> </xsd:restriction> </xsd:simpletype>	すべてのマッピング・レベル: signed char
<xsd:simpleType> <xsd:restriction base="xsd:unsignedByte"> </xsd:restriction> </xsd:simpletype>	すべてのマッピング・レベル: char
<xsd:simpleType> <xsd:restriction base="xsd:short"> </xsd:restriction> </xsd:simpletype>	すべてのマッピング・レベル: short
<xsd:simpleType> <xsd:restriction base="xsd:unsignedShort"> </xsd:restriction> </xsd:simpletype>	すべてのマッピング・レベル: unsigned short
<xsd:simpleType> <xsd:restriction base="xsd:integer"> </xsd:restriction> </xsd:simpletype>	すべてのマッピング・レベル: char[33]
<xsd:simpleType> <xsd:restriction base="xsd:int"> </xsd:restriction> </xsd:simpletype>	すべてのマッピング・レベル: int
<xsd:simpleType> <xsd:restriction base="xsd:unsignedInt"> </xsd:restriction> </xsd:simpletype>	すべてのマッピング・レベル: unsigned int
<xsd:simpleType> <xsd:restriction base="xsd:long"> </xsd:restriction> </xsd:simpletype>	すべてのマッピング・レベル: long long
<xsd:simpleType> <xsd:restriction base="xsd:unsignedLong"> </xsd:restriction> </xsd:simpletype>	すべてのマッピング・レベル: unsigned long long
<xsd:simpleType> <xsd:restriction base="xsd:boolean"> </xsd:restriction> </xsd:simpletype>	すべてのマッピング・レベル: bool (C++ のみ) short (C のみ)
<xsd:simpleType> <xsd:list> <xsd:simpleType> <xsd:restriction base="xsd:int"/> </xsd:simpleType> </xsd:list> </xsd:simpleType>	マッピング・レベル 1.0 の場合 サポートされていない マッピング・レベル 1.1 以上の場合 char[255]
<xsd:simpleType> <xsd:union memberTypes="xsd:int xsd:string"/> </xsd:simpleType>	マッピング・レベル 1.0 の場合 サポートされていない マッピング・レベル 1.1 以上の場合 char[255]

スキーマの **simpleType**	**C** および **C++** のデータ・タイプ
<xsd:simpleType> <xsd:restriction base="xsd:base64Binary"> <xsd:length value=" z"/> </xsd:restriction> </xsd:simpleType> <xsd:simpleType> <xsd:restriction base="xsd:base64binary"> </xsd:restriction> </xsd:simpletype> 長さは定義されていません。	マッピング・レベル 1.1 以下の場合 char[*y*] ここで、*y* =4×(ceil(z/3)) です。 ceil(*x*) は、*x* より大きいか等しい最小の整数です。 マッピング・レベル 1.2 以上の場合 char[*z*] この場合、長さは固定されています。 char[16] 長さが定義されていない場合にバイナリー・データを保管するコンテナーの名前。
<xsd:simpleType> <xsd:restriction base="xsd:float"> </xsd:restriction> </xsd:simpletype>	マッピング・レベル 1.1 以下の場合 char[32] マッピング・レベル 1.2 以上の場合 float(*) **注:** IBM Hexadecimal Floating Point (HFP) データ表記は、XML に使用される IEEE-754-1985 表記とまったく同じではありません。一部の値が 1 つ表記からもう 1 つの表記に正確に変換されない可能性があります。極端に大きいまたは小さい値は、xsd:float データ型に対して有効ではない場合があります。 HFP 表記との間の変換時に、一部の値で精度が失われる可能性があります。正確な変換が重要になる場合、float データ型を使用せずに、固定精度のデータ型を代わりに使用することを検討してください。
<xsd:simpleType> <xsd:restriction base="xsd:double"> </xsd:restriction> </xsd:simpletype>	マッピング・レベル 1.0 以下の場合 char[32] マッピング・レベル 1.2 以上の場合 double(*) **注:** IBM Hexadecimal Floating Point (HFP) データ表記は、XML に使用される IEEE-754-1985 表記とまったく同じではありません。一部の値が 1 つ表記からもう 1 つの表記に正確に変換されない可能性があります。極端に大きいまたは小さい値は、xsd:double データ型に対して有効ではない場合があります。 HFP 表記との間の変換時に、一部の値で精度が失われる可能性があります。正確な変換が重要になる場合、double データ型を使用せずに、固定精度のデータ型を代わりに使用することを検討してください。

PL/I から **XML** スキーマへのマッピング**:**

DFHLS2SC および DFHLS2WS ユーティリティー・プログラムは PL/I データ構造と XML スキーマ定義との間のマッピングをサポートします。 Enterprise PL/I コンパイラーと古い PL/I コンパイラーの間には相違点があるため、PLI-ENTERPRISE と PLI-OTHER の 2 つの言語オプションがサポートされます。

PL/I の名前は、次の規則に従って XML の名前に変換されます。

1. XML エレメント名で無効な文字は、「x」で置き換えられます。

   例えば、monthly$total は monthlyxtotal になります。

2. 重複する名前は、1 つ以上の数字が追加されて固有の名前になります。

   例えば、year の 2 つのインスタンスは year と year1 になります。

DFHLS2SC および DFHLS2WS は次の表に従って PL/I 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 PL/I タイプは DFHLS2SC または DFHLS2WS でサポートされません。 次の制約事項も適用されます。

- COMPLEX 属性を持つデータ項目はサポートされません。
- FLOAT 属性を持つデータ項目は、マッピング・レベル 1.2 以上でサポートされます。 Enterprise PL/I FLOAT IEEE はサポートされません。
- VARYING および VARYINGZ の純粋な DBCS ストリングは、マッピング・レベル 1.2 以上でサポートされます。
- 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。
- PICTURE ストリングはサポートされません。
- ORDINAL データ項目は、FIXED BINARY(7) データ・タイプとして扱われます。
- **MAPPING-LEVEL** パラメーターを 1.2 以上に設定して、**CHAR-VARYING** パラメーターを NULL に設定すると、文字配列は `xsd:string` にマップされ、ヌル終了ストリングとして処理されます。このマッピングは Enterprise PL/I には適用されません。
- **MAPPING-LEVEL** パラメーターを 1.2 以上に設定して、**CHAR-VARYING** パラメーターを BINARY に設定すると、文字配列は `xsd:base64Binary` にマップされ、バイナリー・データとして処理されます。
- **MAPPING-LEVEL** パラメーターを 1.2 以上に設定して、**CHAR-VARYING** パラメーターを COLLAPSE に設定すると、`<xsd:whiteSpace value="collapse"/>` がストリング用に生成されます。

DFHLS2SC および DFHLS2WS は、PL/I の埋め込みアルゴリズムを完全には実装しないため、データ構造で埋め込みバイトを明示的に宣言する必要があります。 DFHLS2SC および DFHLS2WS は、埋め込みバイトがないことを検出すると、メッセージを発行します。 それぞれの最上位構造は、ダブルワード境界で開始して、構造内のそれぞれのバイトは正しい境界にマップされている必要があります。 次のコード・フラグメントについて考えてみます。

```
 3 FIELD1 FIXED BINARY(7),
3 FIELD2 FIXED BINARY(31),
3 FIELD3 FIXED BINARY(63);
```

この例では以下のようになります。

- FIELD1 の長さは 1 バイトで、任意の境界に合わせることができます。
- FIELD2 の長さは 4 バイトで、フルワード境界に合わせる必要があります。
- FIELD3 の長さは 8 バイトで、ダブルワード境界に合わせる必要があります。

Enterprise PL/I コンパイラーはフィールドを以下の順序に調整します。

1. FIELD3 の境界要件が最も厳しいため、FIELD3 が最初に配置されます。
2. FIELD2 は FIELD3 の直前のフルワード境界に合わせます。
3. FIELD1 が FIELD3 の直前のバイト境界に合わせます。

最後に、構造全体がフルワード境界で位置合わせされるように、コンパイラーは FIELD1 の直前に 3 つの埋め込みバイトを挿入します。

DFHLS2WS は同等の埋め込みバイトを挿入しないため、DFHLS2WS が構造を処理する前にこれらの埋め込みバイトを明示的に宣言する必要があります。以下に例を示します。

```
 3 PAD1 FIXED BINARY(7),
3 PAD2 FIXED BINARY(7),
3 PAD3 FIXED BINARY(7),
3 FIELD1 FIXED BINARY(7),
3 FIELD2 FIXED BINARY(31),
3 FIELD3 FIXED BINARY(63);
```

あるいは、すべてのフィールドを位置合わせされないと宣言するよう構造を変更して、この構造を使用するアプリケーションを再コンパイルすることもできます。PL/I 構造上のメモリー位置合わせ要件について詳しくは、Enterprise PL/I for z/OS の製品情報を参照してください。

PL/I のデータ記述	スキーマ

`FIXED BINARY (`*n*`)`
 	
ここで *n* ≤ 7
	`<xsd:simpleType>`
	`<xsd:restriction base="xsd:byte"/>`
	`</xsd:simpleType>`

`FIXED BINARY (`*n*`)`
 	
ここで 8 ≤ *n* ≤ 15
	`<xsd:simpleType>`
	`<xsd:restriction base="xsd:short"/>`
	`</xsd:simpleType>`

`FIXED BINARY (`*n*`)`
 	
ここで 16 ≤ *n* ≤ 31
	`<xsd:simpleType>`
	`<xsd:restriction base="xsd:int"/>`
	`</xsd:simpleType>`

`FIXED BINARY (`*n*`)`
 	
ここで 32 ≤ *n* ≤ 63
制約事項: Enterprise PL/I のみ
	`<xsd:simpleType>`
	`<xsd:restriction base="xsd:long"/>`
	`</xsd:simpleType>`

`UNSIGNED FIXED BINARY(`*n*`)`
 	
ここで *n* ≤ 8
制約事項: Enterprise PL/I のみ
	`<xsd:simpleType>`
	`<xsd:restriction base="xsd:unsignedByte"/>`
	`</xsd:simpleType>`

`UNSIGNED FIXED BINARY(`*n*`)`
 	
ここで 9 ≤ *n* ≤ 16
制約事項: Enterprise PL/I のみ
	`<xsd:simpleType>`
	`<xsd:restriction base="xsd:unsignedShort"/>`
	`</xsd:simpleType>`

`UNSIGNED FIXED BINARY(`*n*`)`
 	
ここで 17 ≤ *n* ≤ 32
制約事項: Enterprise PL/I のみ
	`<xsd:simpleType>`
	`<xsd:restriction base="xsd:unsignedInt"/>`
	`</xsd:simpleType>`

`UNSIGNED FIXED BINARY(`*n*`)`
 	
ここで 33 ≤ *n* ≤ 64
制約事項: Enterprise PL/I のみ
	`<xsd:simpleType>`
	`<xsd:restriction base="xsd:unsignedLong"/>`
	`</xsd:simpleType>`

PL/I のデータ記述	スキーマ
FIXED DECIMAL(*n*,*m*)	<xsd:simpleType> <xsd:restriction base="xsd:decimal"> <xsd:totalDigits value=" *n*"/> <xsd:fractionDigits value=" *m*"/> </xsd:restriction> </xsd:simpleType>
FIXED DECIMAL(15) DATETIME=PACKED15 の場合マッピング・レベル 3.0 以上でサポートされる	<xsd:simpleType> <xsd:restriction base="xsd:dateTime" </xsd:restriction> </xsd:simpleType> タイム・スタンプの形式は CICS ABSTIME です。
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* /8
CHARACTER(*n*) VARYING および VARYINGZ はマッピング・レベル 1.2 以上でもサポートされる **制約事項:** VARYINGZ は Enterprise PL/I でのみサポートされる	<xsd:simpleType> <xsd:restriction base="xsd:string"> <xsd:maxLength value=" *n*"/> <xsd:whiteSpace value="preserve"/> </xsd:restriction> </xsd:simpleType>
GRAPHIC(*n*) VARYING および VARYINGZ はマッピング・レベル 1.2 以上でもサポートされる **制約事項:** VARYINGZ は Enterprise PL/I でのみサポートされる	マッピング・レベル 1.0 および 1.1 で、 *m* = 2**n* の場合: <xsd:simpleType> <xsd:restriction base="xsd:hexBinary"> <xsd:length value=" *m*"/> </xsd:restriction> </xsd:simpleType> マッピング・レベル 1.2 以上の場合: <xsd:simpleType> <xsd:restriction base="xsd:string"> <xsd:length value=" *n*"/> <xsd:whiteSpace value="preserve"/> </xsd:restriction> </xsd:simpleType>

PL/I のデータ記述	スキーマ
WIDECHAR(*n*) **制約事項:** Enterprise PL/I のみ	マッピング・レベル 1.0 および 1.1 で、 *m* = 2*n* の場合: <xsd:simpleType> <xsd:restriction base="xsd:hexBinary"> <xsd:length value=" *m*"/> </xsd:restriction> </xsd:simpleType> マッピング・レベル 1.2 以上の場合: <xsd:simpleType> <xsd:restriction base="xsd:hexBinary"> <xsd:length value=" *n*"/> </xsd:restriction> </xsd:simpleType> マッピング・レベル 4.0 以上では、CICS はアプリケーション・データ構造フィールドに UTF-16 データを取り込みます。 <xsd:simpleType> <xsd:restriction base="xsd:string"> <xsd:maxlength value="n"/> <xsd:whiteSpace value="preserve"/> </xsd:restriction> </xsd:simpleType>
ORDINAL **制約事項:** Enterprise PL/I のみ	<xsd:simpleType> <xsd:restriction base="xsd:byte"/> </xsd:simpleType>
BINARY FLOAT(*n*) ここで、*n* <= 21 マッピング・レベル 1.2 以上でサポートされる	<xsd:simpleType> <xsd:restriction base="xsd:float"> </xsd:restriction> </xsd:simpletype>
BINARY FLOAT(*n*) ここで、21 < *n* <= 53 53 より大きい値はサポートされない。 マッピング・レベル 1.2 以上でサポートされる	<xsd:simpleType> <xsd:restriction base="xsd:double"> </xsd:restriction> </xsd:simpletype>

PL/I のデータ記述	スキーマ
DECIMAL FLOAT(*n*) ここで、*n* <= 6 です。 マッピング・レベル 1.2 以上でサポートされる **注:** IBM Hexadecimal Floating Point (HFP) データ表記は、XML に使用される IEEE-754-1985 表記とまったく同じではありません。一部の値が 1 つ表記からもう 1 つの表記に正確に変換されない可能性があります。極端に大きいまたは小さい値は、xsd:float データ型に対して有効ではない場合があります。 HFP 表記との間の変換時に、一部の値で精度が失われる可能性があります。正確な変換が重要になる場合、DECIMAL FLOAT データ型を使用せずに、固定精度のデータ型を代わりに使用することを検討してください。	<xsd:simpleType> <xsd:restriction base="xsd:float"> </xsd:restriction> </xsd:simpletype>
DECIMAL FLOAT(*n*) ここで、6 < *n* <= 16 16 より大きい値はサポートされない。 マッピング・レベル 1.2 以上でサポートされる **注:** IBM Hexadecimal Floating Point (HFP) データ表記は、XML に使用される IEEE-754-1985 表記とまったく同じではありません。一部の値が 1 つ表記からもう 1 つの表記に正確に変換されない可能性があります。極端に大きいまたは小さい値は、xsd:double データ型に対して有効ではない場合があります。 HFP 表記との間の変換時に、一部の値で精度が失われる可能性があります。正確な変換が重要になる場合、DECIMAL FLOAT データ型を使用せずに、固定精度のデータ型を代わりに使用することを検討してください。	<xsd:simpleType> <xsd:restriction base="xsd:double"> </xsd:restriction> </xsd:simpletype>

XML スキーマから **PL/I** へのマッピング**:**

DFHSC2LS および DFHWS2LS ユーティリティー・プログラムは XML スキーマ定義と PL/I データ構造の間のマッピングをサポートします。 Enterprise PL/I コンパイラーと古い PL/I コンパイラーの間には相違点があるため、PLI-ENTERPRISE と PLI-OTHER の 2 つの言語オプションがサポートされます。

PL/I にスキーマ・エレメント名をマップする場合の規則

CICS 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 PL/I 変数の固有で有効な名前を生成します。

1. A から Z、a から z、0 から 9、@、#、_、または $ 以外の文字は、「X」で置き換えられます。

   例えば、 monthly-total は monthlyXtotal となります。

MAPPING-OVERRIDES パラメーターを使用して、他の文字の処理方法を変更できます。 例えば、値 **HYPHENS-AS-UNDERSCORES** を設定すると、XML に含まれるハイフンは X ではなく下線に変換されます。例えば、monthly-total は monthly_total となります。

2. 変数の基数が可変であることがスキーマで指定されている (つまり、minOccurs 属性および maxOccurs 属性が xsd:element で異なる値によって指定されている) 場合、24 文字より長いスキーマのエレメント名は、24 文字に切り捨てられます。

   変数の基数が固定であることがスキーマで指定されている場合、29 文字より長いスキーマのエレメント名は、29 文字に切り捨てられます。

3. 同じスコープ内の重複した名前は、名前の 2 番目以降のインスタンスに 1 つ以上の数字を追加することによって固有にします。

   例えば、 year の 3 つのインスタンスは year 、 year1 、および year2 になります。

   もし上記の動作が望ましくないなら、ユーザーはユーティリティーの入力として MAPPING-OVERRIDES=NO-ARRAY-NAME-INDEXING を指定できます。これにより、名前の 2 番目以降のインスタンスへの 1 つまたは 2 つの数字の追加が無効になります。

4. _cont または _num ストリング用に 5 つの文字が予約されており、変数の基数が可変であることがスキーマで指定されている (つまり、minOccurs 属性と maxOccurs 属性が異なる値で指定されている) 場合に使用されます。

   詳しくは、 389 ページの『変化するエレメントの配列』を参照してください。

5. 属性では、前の規則がエレメント名に適用されます。 エレメント名に接頭部 attr- が追加され、その後に -value または -exist が続きます。全長が 28 文字を超える場合、エレメント名は切り捨てられます。 詳しくは、 394 ページの『XML 属性のサポート』を参照してください。

   nillable 属性には特別な規則があります。接頭部 attr- が追加されますが、エレメント名の先頭には nil- も追加されます。 エレメント名の後には -value が続きます。 全長が 28 文字を超える場合、エレメント名は切り捨てられます。

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。

PL/I にスキーマ・タイプをマップする場合の規則

DFHSC2LS および DFHWS2LS は、次の表に従ってスキーマ・タイプを PL/I のデータ・タイプにマップします。以下の点にも注意してください。

- **MAPPING-LEVEL** パラメーターを 1.2 以上に設定して、**CHAR-VARYING** パラメーターを NULL に設定すると、可変長文字データはヌル終了ストリングにマップされ、ヌル終止符として追加された 1 つの文字が割り振られます。
- **MAPPING-LEVEL** パラメーターを 1.2 以上に設定して、**CHAR-VARYING** パラメーターを指定しないと、可変長文字データはデフォルトでは、Enterprise PL/I の場合は VARYINGZ データ・タイプにマップされ、その他の PL/I の場合は VARYING データ・タイプにマップされます。

- 可変長バイナリー・データは、32 768 バイトより少ない場合は VARYING データ・タイプにマップされ、32 768 バイトを超える場合はコンテナーにマップされます。

スキーマ	**PL/I** のデータ記述
<xsd:simpleType> <xsd:restriction base="xsd:anyType"> </xsd:restriction> </xsd:simpleType>	マッピング・レベル 2.0 以下の場合 サポートされていない マッピング・レベル 2.1 以上の場合 Supported
<xsd:simpleType> <xsd:restriction base="xsd:anySimpletype"> </xsd:restriction> </xsd:simpleType>	マッピング・レベル 1.1 以上の場合 CHAR(255)
<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	すべてのマッピング・レベル CHARACTER(*z*)
<xsd:simpleType> <xsd:restriction base="xsd: *type*"> <xsd: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	マッピング・レベル 4.0 以上で **CCSID=1200** の場合: WIDECHAR(*z*)

スキーマ	**PL/I** のデータ記述
<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 time gDay gMonth gYear gMonthDay gYearMonth	すべてのマッピング・レベル CHAR(32)
<xsd:simpleType> <xsd:restriction base="xsd:dateTime"> </xsd:restriction> </xsd:simpleType>	マッピング・レベル 1.2 以下の場合 CHAR(32) マッピング・レベル 2.0 以上の場合 CHAR(40) マッピング・レベル 3.0 以上の場合 FIXED DECIMAL(15) タイム・スタンプのフォーマットは CICS ABSTIME です。
<xsd:simpleType> <xsd:restriction base="xsd:hexBinary"> <xsd:length value=" *y*"/> </xsd:restriction> </xsd:simpleType>	マッピング・レベル 1.1 以下の場合 BIT(*z*) ここで、 $z = 8 \times y$ かつ $z < 4095$ バイト。 CHAR(*z*) ここで、 $z = 8 \times y$ かつ $z > 4095$ バイト。 マッピング・レベル 1.2 以上: CHAR(*y*)

スキーマ	**PL/I** のデータ記述
<xsd:simpleType> <xsd:restriction base="xsd:byte"> </xsd:restriction> </xsd:simpleType>	すべてのマッピング・レベル: **Enterprise PL/I** SIGNED FIXED BINARY (7) その他の **PL/I** FIXED BINARY (7)
<xsd:simpleType> <xsd:restriction base="xsd:unsignedByte"> </xsd:restriction> </xsd:simpleType>	すべてのマッピング・レベル: **Enterprise PL/I** UNSIGNED FIXED BINARY (8) その他の **PL/I** FIXED BINARY (8)
<xsd:simpleType> <xsd:restriction base="xsd:short"> </xsd:restriction> </xsd:simpleType>	すべてのマッピング・レベル: **Enterprise PL/I** SIGNED FIXED BINARY (15) その他の **PL/I** FIXED BINARY (15)
<xsd:simpleType> <xsd:restriction base="xsd:unsignedShort"> </xsd:restriction> </xsd:simpleType>	すべてのマッピング・レベル: **Enterprise PL/I** UNSIGNED FIXED BINARY (16) その他の **PL/I** FIXED BINARY (16)
<xsd:simpleType> <xsd:restriction base="xsd:integer"> </xsd:restriction> </xsd:simpleType>	すべてのマッピング・レベル: **Enterprise PL/I** FIXED DECIMAL(31,0) その他の **PL/I** FIXED DECIMAL (15,0)
<xsd:simpleType> <xsd:restriction base="xsd:int"> </xsd:restriction> </xsd:simpleType>	すべてのマッピング・レベル: **Enterprise PL/I** SIGNED FIXED BINARY (31) その他の **PL/I** FIXED BINARY (31)
<xsd:simpleType> <xsd:restriction base="xsd:unsignedInt"> </xsd:restriction> </xsd:simpleType>	すべてのマッピング・レベル: **Enterprise PL/I** UNSIGNED FIXED BINARY(32) その他の **PL/I** BIT(64)

スキーマ	**PL/I** のデータ記述
<xsd:simpleType> <xsd:restriction base="xsd:long"> </xsd:restriction> </xsd:simpleType>	マッピング・レベル 1.1 以下の場合 **Enterprise PL/I** SIGNED FIXED BINARY(63) **注:** LIMITS コンパイラー・ディレクティブは、PL/I コンパイラーによるこのフィールドの解釈方法に影響する場合があります。CICS は、このフィールドが宣言されたサイズであることを想定しますが、コンパイラーはこのフィールドをより小さなスペースに最適化することがあるため、不一致が生じる可能性があります。このような問題を回避するには、LIMITS(FIXEDBIN(63)) コンパイル時オプションを使用します。 マッピング・レベル 1.2 以上の場合 **Enterprise PL/I** CHAR(*y*) ここで *y* は 16 MB より小さい固定長です。 すべてのマッピング・レベル: その他の **PL/I** BIT(64)
<xsd:simpleType> <xsd:restriction base="xsd:unsignedLong"> </xsd:restriction> </xsd:simpleType>	マッピング・レベル 1.1 以下の場合 **Enterprise PL/I** UNSIGNED FIXED BINARY(64) **注:** LIMITS コンパイラー・ディレクティブは、PL/I コンパイラーによるこのフィールドの解釈方法に影響する場合があります。CICS は、このフィールドが宣言されたサイズであることを想定しますが、コンパイラーはこのフィールドをより小さなスペースに最適化することがあるため、不一致が生じる可能性があります。このような問題を回避するには、LIMITS(FIXEDBIN(63)) コンパイル時オプションを使用します。 マッピング・レベル 1.2 以上の場合 **Enterprise PL/I** CHAR(*y*) ここで *y* は 16 MB より小さい固定長です。 すべてのマッピング・レベル: その他の **PL/I** BIT(64)

スキーマ	**PL/I** のデータ記述
<xsd:simpleType> <xsd:restriction base="xsd:boolean"> </xsd:restriction> </xsd:simpleType>	マッピング・レベル 1.1 以下の場合 **Enterprise PL/I** SIGNED FIXED BINARY (7) その他の **PL/I** FIXED BINARY (7) マッピング・レベル 1.2 以上の場合 **Enterprise PL/I** BIT(7) BIT(1) その他の **PL/I** BIT(7) BIT(1) ここで、BIT(7) は位置合わせのために提供されており、BIT(1) にはブールでマップされた値が含まれます。
<xsd:simpleType> <xsd:restriction base="xsd:decimal"> <xsd:totalDigits value=" *n*"/> <xsd:fractionDigits value=" *m*"/> </xsd:restriction> </xsd:simpleType>	すべてのマッピング・レベル FIXED DECIMAL(*n* , *m*)
<xsd:simpleType> <xsd:list> <xsd:simpleType> <xsd:restriction base="xsd:int"/> </xsd:simpleType> </xsd:list> </xsd:simpleType>	すべてのマッピング・レベル CHAR(255)
<xsd:simpleType> <xsd:union memberTypes="xsd:int xsd:string"/> </xsd:simpleType>	すべてのマッピング・レベル CHAR(255)

スキーマ	**PL/I** のデータ記述
<xsd:simpleType> <xsd:restriction base="xsd:base64Binary"> <xsd:length value=" *y*"/> </xsd:restriction> </xsd:simpleType> <xsd:simpleType> <xsd:restriction base="xsd:base64Binary"> </xsd:restriction> </xsd:simpleType> 長さは定義されていません。	マッピング・レベル 1.0 の場合 サポートされていない マッピング・レベル 1.1 の場合 CHAR(*z*) ここで、*z* =4×(ceil(*y* /3)) です。ceil(*x*) は、*x* より大きいか等しい最小の整数です。 マッピング・レベル 1.2 以上の場合 CHAR(*y*) この場合、長さは固定されています。 CHAR(16) この場合、長さは定義されていません。 このフィールドは、2 進データが保管されるコンテナーの 16 バイトの名前を保持します。
<xsd:simpleType> <xsd:restriction base="xsd:float"> </xsd:restriction> </xsd:simpletype>	マッピング・レベル 1.0 および 1.1 の場合 CHAR(32) マッピング・レベル 1.2 以上の場合 **Enterprise PL/I** DECIMAL FLOAT(6) HEXADEC その他の **PL/I** DECIMAL FLOAT(6) **注:** IBM Hexadecimal Floating Point (HFP) データ表記は、XML に使用される IEEE-754-1985 表記とまったく同じではありません。一部の値が 1 つ表記からもう 1 つの表記に正確に変換されない可能性があります。極端に大きいまたは小さい値は、xsd:float データ型に対して有効ではない場合があります。 HFP 表記との間の変換時に、一部の値で精度が失われる可能性があります。正確な変換が重要になる場合、DECIMAL FLOAT データ型を使用せずに、固定精度のデータ型を代わりに使用することを検討してください。

スキーマ	**PL/I** のデータ記述
<xsd:simpleType> <xsd:restriction base="xsd:double"> </xsd:restriction> </xsd:simpletype>	マッピング・レベル 1.0 および 1.1 の場合 CHAR(32) マッピング・レベル 1.2 以上の場合 **Enterprise PL/I** DECIMAL FLOAT(16) HEXADEC その他の **PL/I** DECIMAL FLOAT(16) **注:** IBM Hexadecimal Floating Point (HFP) データ表記は、XML に使用される IEEE-754-1985 表記とまったく同じではありません。一部の値が 1 つ表記からもう 1 つの表記に正確に変換されない可能性があります。極端に大きいまたは小さい値は、xsd:double データ型に対して有効ではない場合があります。 HFP 表記との間の変換時に、一部の値で精度が失われる可能性があります。正確な変換が重要になる場合、DECIMAL FLOAT データ型を使用せずに、固定精度のデータ型を代わりに使用することを検討してください。

変化するエレメントの配列**:**

XML は、エレメント数が変化する配列を持つことができます。一般に、エレメント数が変化する WSDL 文書や XML スキーマは 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。 CICS はコンテナー・ベースのマッピングまたはインライン・マッピングを使用して XML におけるエレメント数の変化に対応します。

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。

- 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。

これは、アプリケーションの XML または 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。

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

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:length value="8"/>
</xsd:restriction>
</xsd:simpleType>
</xsd:element>
```

一般に、エレメント数が変化する WSDL 文書は 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 は一連のコンテナーとしてアプリケーション・プログラム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。これらの構造はアプリケーションへの入出力として使用されます。

- CICS が XML をアプリケーション・データに変換する場合、これらの構造にアプリケーション・データが追加され、アプリケーションがそのデータを読み取ります。
- CICS がアプリケーション・データを XML に変換する場合、アプリケーションによって構造内に追加されたアプリケーション・データが読み取られます。

これらのデータ構造のフォーマットは、一連の例を使用すると分かりやすくなります。 SOAP メッセージまたはアプリケーションの XML を使用できます。 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:length 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
```

マッピング・レベル **2** 以下における変化するエレメント数

この例では、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:length value="8"/>
</xsd:restriction>
</xsd:simpleType>
</xsd:element>
```

主データ構造には、2 つのフィールドの宣言が格納されます。 CICS が XML をバイナリー・データに変換すると、最初のフィールドである component-num にはエレメントが XML 内に現れる回数が格納され、2 番目のフィールドである component-cont には、コンテナーの名前が格納されます。

```
05 component-num PIC S9(9) COMP-5
05 component-cont PIC X(16)
```

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)
```

エレメントが発生する回数を割り出すには、component-num (1 から 5 までの範囲の値が入ります) の値を検査する必要があります。 エレメント・コンテンツは component-cont で指定されたコンテナー内にあります。このコンテナーはエレメントの配列を保持しており、各エレメントは DFHWS-component データ構造によってマップされています。

minOccurs="0" および maxOccurs="1" である場合、このエレメントはオプションです。 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、以下のように component-num 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 エレメントはなく、component-cont の内容は未定義です。
- この値が 1 の場合、component エレメントは component-cont 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。

注**:** SOAP メッセージが単一の繰り返しエレメントで構成される場合、DFHWS2LS は 2 つの言語構造を生成します。 主要な言語構造は、配列エレメントの数と、エレメントの配列を保持するコンテナーの名前を含んでいます。 2 番目の言語構造は、繰り返しエレメントの単一インスタンスをマップします。

マッピング・レベル **2.1** 以上における変化するエレメント数

マッピング・レベル 2.1 以上では、CICS アシスタントで **INLINE-MAXOCCURS-LIMIT** パラメーターを使用できます。 **INLINE-MAXOCCURS-LIMIT** パラメーターは変化するエレメント数を処理する方法を指定します。 可変数のエレメント用のマッピング・オプションは、コンテナー・ベースのマッピング (390 ページの『マッピング・レベル 2 以下における変化するエレメント数』で説明されています) またはインライン・マッピングです。 このパラメーターの *value* は、0 - 32767 までの範囲の正整数です。

- **INLINE-MAXOCCURS-LIMIT** のデフォルト値は 1 であり、オプション・エレメントがインラインで確実にマップされます。
- **INLINE-MAXOCCURS-LIMIT** パラメーターに値 0 が設定されている場合、インライン・マッピングは使用されません。
- maxOccurs が **INLINE-MAXOCCURS-LIMIT** の値以下の場合、インライン・マッピングが使用されます。
- maxOccurs が **INLINE-MAXOCCURS-LIMIT** の値以上の場合、コンテナー・ベースのマッピングが使用されます。

変化するエレメント数をインラインでマップすると、配列 (上記の例では発生するオカレンスが固定) およびカウンターが生成されます。 component-num フィールドは存在するエレメントのインスタンス数を示し、それらのインスタンスは配列によって示されます。 390 ページの『マッピング・レベル 2 以下における変化するエレメント数』で示される例では、**INLINE-MAXOCCURS-LIMIT** が 5 以下の場合に生成されるデータ構造は次のようになります。

```
05 component-num PIC S9(9) COMP-5 SYNC.
05 component OCCURS 5 PIC X(8).
```

最初のフィールド (component-num) は前のセクションで示されたコンテナー・ベース・マッピングの例の出力と同じです。 2 番目のフィールドには長さが 5 の配列があり、生成される可能性のある最大エレメント数を収容できる大きさです。

インライン・マッピングは、エレメントの出現回数およびデータが収容されるコンテナーの名前を格納するコンテナー・ベースのマッピングとは異なり、現行のコンテナーにあるすべてのデータを格納します。 現行のコンテナーにデータを格納するとパフォーマンスが向上するので、インライン・マッピングの方が一般的には望ましいといえます。

ネストされた変数配列

複雑な WSDL 文書および XML スキーマには可変の繰り返しエレメントを含めることができ、そのエレメントにはさらに可変の繰り返しエレメントを含めることができます。 この場合、記述される構造は、例で説明した 2 つのレベルを超えます。

この例は、1 回から 5 回出現する可能性がある必須エレメント (<component1>) 内でネストされたオプションのエレメント (<component2>) を示します。

```
<xsd:element name="component1"
minOccurs="1" maxOccurs="5">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="component2"
minOccurs="0" maxOccurs="1">
<xsd:simpleType>
<xsd:restriction base="xsd:string">
<xsd:length value="8"/>
</xsd:restriction>
</xsd:simpleType>
</xsd:element>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
```

最上位のデータ構造は、前の例のデータ構造とまったく同じです。

```
05 component1-num PIC S9(9) COMP-5
05 component1-cont PIC X(16)
```

ただし、2 番目のデータ構造には、以下のエレメントが格納されます。

```
01 DFHWS-component1
02 component2-num PIC S9(9) COMP-5
02 component2-cont PIC X(16)
```

3 番目の構造には以下のエレメントが格納されます。

```
01 DFHWS-component2
02 component2 PIC X(8)
```

最外部エレメント <component1> の出現回数は、component1-num に格納されます。

component1-cont で指定されたコンテナーには、2 番目のデータ構造 (DFHWS-component1) のその数のインスタンスを持つ配列が含まれています。

component2-cont の各インスタンスは、異なるコンテナーを指定します。それぞれ、第 3 レベルの構造 (DFHWS-component2) によってマップされたデータ構造が含まれています。

この構造を説明するために、例と一致する次のような XML のフラグメントについて考えてみます。

```
<component1><component2>
string1
</component2></component1>
<component1><component2>
string2
</component2></component1>
<component1></component1>
```

<component1> は 3 回発生します。 最初の 2 つには、それぞれ <component2> のインスタンスが含まれていますが、3 番目のインスタンスには含まれていません。

最上位のデータ構造では、component1-num に値 3 が含まれています。 component1-cont で指定されたコンテナーには、DFHWS-component1 の次の 3 つのインスタンスがあります。

1. 第 1 のインスタンスでは、component2-num の値は 1 で、component2-cont で指定されたコンテナーは *string1* を保持します。
2. 第 2 のインスタンスでは、component2-num の値は 1 で、component2-cont で指定されたコンテナーは *string2* を保持します。
3. 第 3 のインスタンスでは、component2-num の値は 0 で、component2-cont の内容は未定義です。

このインスタンスでは、完全なデータ構造は、次の合計 4 つのコンテナーによって表現されます。

- コンテナー DFHWS-DATA 内のルート・データ構造
- component1-cont で指定されたコンテナー
- component2-cont の最初の 2 つのインスタンスで指定された 2 つのコンテナー

オプションの構造および xsd:choice

DFHWS2LS および DFHSC2LS はマッピング・レベル 2.1 以上で、<xsd:sequence>、<xsd:choice>、および <xsd:all> エレメントでの maxOccurs および minOccurs の使用をサポートします。この場合、minOccurs 属性および maxOccurs 属性は minOccurs="0" および maxOccurs="1" に設定されています。

アシスタントは、エレメント内の子エレメントがそれぞれオプションであるかのようにしてエレメントを処理するマッピングを生成します。 これらのエレメントを使用してアプリケーションを実装する場合、アプリケーションが無効なオプションの

組み合わせを生成しないように確認してください。 それぞれのエレメントの生成された言語構造には count フィールドがありますが、これらのすべてが「0」に設定されるか、すべてが「1」に設定される必要があります。これ以外の値の組み合わせは、<xsd:choice> エレメントの場合を除き、無効です。

<xsd:choice> エレメントは、そのエレメントのオプションが 1 つのみ使用できることを示します。 これはすべてのマッピング・レベルでサポートされます。 アシスタントは <xsd:choice> のそれぞれのオプションを、minOccurs="0" および maxOccurs="1" である <xsd:sequence> エレメントのように扱います。
<xsd:choice> エレメントを使用してアプリケーションを実装する場合は、アプリケーションが無効なオプションの組み合わせを生成しないように確認してください。 それぞれのエレメントの生成された言語構造には count フィールドがありますが、そのうちの 1 つが「1」に、他のすべてが「0」に設定される必要があります。 これ以外の組み合わせはすべて無効です。例外は、<xsd:choice> のエレメント自体がオプションの場合であり、すべてのフィールドが 0 に設定されていても有効です。

XML 属性のサポート**:**

XML スキーマは、XML で許可される属性または必須の属性を指定することができます。 CICS アシスタント・ユーティリティーである DFHWS2LS および DFHSC2LS はデフォルトで XML 属性を無視します。 XML スキーマに定義されている XML 属性を処理するには、**MAPPING-LEVEL** パラメーターの値が 1.1 以上である必要があります。

オプションの属性

属性にはオプションの属性と必須属性があり、SOAP メッセージ内またはアプリケーションの XML 内の任意のエレメントに関連付けることができます。 スキーマで定義されたすべてのオプションの属性ごとに、次に示す 2 つのフィールドが適切な言語構造で生成されます。

1. 存在フラグ。このフィールドはブール・データ・タイプとして扱われ、通常は長さが 1 バイトです。
2. 値。このフィールドは、同等のタイプの XML エレメントと同じ方法でマップされます。 例えば、タイプ NMTOKEN の属性は、タイプ NMTOKEN の XML エレメントと同じようにマップされます。

属性の存在フィールドと値フィールドは、関連付けられたエレメントのフィールドの前に、生成された言語構造で表示されます。 インスタンス文書に現れる予期しない属性は無視されます。

例えば、次のスキーマ属性定義について考えてみます。

```
<xsd:attribute name="age" type="xsd:short"
use="optional" />
```

このオプションの属性は次に示す COBOL 構造にマップされます。

```
05 attr-age-exist PIC X DISPLAY
05 attr-age-value PIC S9999 COMP-5 SYNC
```

オプションの属性のランタイム処理

オプションの属性では次に示すランタイム処理が実行されます。

- 属性が存在する場合は、存在フラグが設定され、値はマップされます。
- 属性が存在しない場合は、存在フラグは設定されません。
- 属性がデフォルト値を持ち、存在する場合は、値がマップされます。
- 属性がデフォルト値を持ち、存在しない場合は、デフォルト値がマップされます。

デフォルト値を持つオプションの属性は、必須属性として扱われます。

CICS がデータを XML に変換すると、次のランタイム処理が行われます。

- 存在フラグが設定される場合は、属性は変換されて XML に含められます。
- 存在フラグが設定されない場合は、属性は XML に含められません。

必須属性とランタイム処理

すべての必須属性で、値フィールドのみが適切な言語構造で生成されます。

属性が XML に存在する場合は、値がマップされます。 属性が存在しない場合、次に示す処理が行われます。

- アプリケーションが Web サービス・プロバイダーの場合、CICS はクライアントの SOAP メッセージ内に、エラーを示す SOAP 障害メッセージを生成します。
- アプリケーションが Web サービス要求側の場合、CICS はメッセージを発行して、13 の RESP2 コード付きの変換エラー応答をアプリケーションに返します。
- アプリケーションが **TRANSFORM XMLTODATA** コマンドを使用している場合、CICS はメッセージを発行し、無効な要求応答 (RESP2 コード 3) をアプリケーションに戻します。

CICS が COMMAREA またはコンテナーの内容を基にして SOAP メッセージを生成すると、属性は変換されて、メッセージに含められます。 アプリケーションが **TRANSFORM DATATOXML** コマンドを使用する場合、CICS は属性の変換も行って XML に含めます。

nillable 属性

nillable 属性とは、XML スキーマの `xsd:element` で指定される場合のある特殊な属性です。これは、XML のエレメントで `xsi:nil` 属性が有効であることを指定します。エレメントで `xsi:nil` 属性が指定されている場合、エレメントは存在するが値がないため、このエレメントには内容が関連付けられていないことを示します。

XML スキーマが nillable 属性を true として定義した場合は、ブール値を使用する必須属性としてマップされます。

CICS が `xsi:nil` 属性を含む SOAP メッセージを受信した場合や、この属性を含む XML をアプリケーション用に変換する必要がある場合、属性の値は true または false です。値が true の場合は、`xsi:nil` 属性のスコープ内のエレメントまたはネストされたエレメントは、アプリケーションによって無視される必要があります。

CICS が、xsi:nil 属性の値が true である COMMAREA またはコンテナーの内容を基にして SOAP メッセージまたは XML を生成すると、次に示す処理が行われます。

- xsi:nil 属性が XML または SOAP メッセージに生成されます。
- 関連するエレメントの値は無視されます。
- エレメント内でネストされたエレメントはすべて無視されます。

SOAP メッセージの例

WSDL 文書の一部になる場合がある次の例の XML スキーマについて考えてみます。

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<xsd:element name="root" nillable="true">
<xsd:complexType>
<xsd:sequence>
<xsd:element nillable="true" name="num" type="xsd:int" maxOccurs="3"
minOccurs="3"/>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:schema>
```

このスキーマに適合する部分的な SOAP メッセージの例を、次に示します。

```
<root xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<num xsi:nil="true"/>
<num>15</num>
<num xsi:nil="true"/>
</root>
```

COBOL では、この SOAP メッセージは次のエレメントにマップされます。

```
05 root
10 attr-nil-root-value PIC X DISPLAY
10 num OCCURS 3
15 num1 PIC S9(9) COMP-5 SYNC
15 attr-nil-num-value PIC X DISPLAY
10 filler PIC X(3)
```

<xsd:any> および **xsd:anyType** のサポート:

DFHWS2LS および DFHSC2LS は XML スキーマにおける <xsd:any> および xsd:anyType の使用をサポートしています。 <xsd:any> XML スキーマ・エレメントを使用して未定義の内容を持つ XML 文書のセクションを記述できます。 xsd:anyType とは、すべての単純および複合データ・タイプの派生元となる基本のデータ・タイプです。データ内容に制限や制約はありません。

CICS アシスタントで <xsd:any> および xsd:anyType を使用する前に、次に示すパラメーターを設定してください。

- **MAPPING-LEVEL** パラメーターを 2.1 以上に設定します。
- Web サービス・プロバイダー・アプリケーションの場合、**PGMINT** パラメーターを CHANNEL に設定します。

<xsd:any> の例

この例では <xsd:any> エレメントを使用して、"Customer" タグの "Surname" タグの後に、オプションの非構造化 XML コンテンツを記述しています。

```
<xsd:element name="Customer">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="Title" type="xsd:string"/>
<xsd:element name="FirstName" type="xsd:string"/>
<xsd:element name="Surname" type="xsd:string"/>
<xsd:any minOccurs="0"/>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
```

この XML スキーマに適合する SOAP メッセージの例を以下に示します。

```
<xml version='1.0' encoding='UTF-8'?>
    <SOAP-ENV:Envelope
    xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Body>
    <Customer xmlns="http://www.example.org/anyExample">
    <Title xmlns="">Mr</Title>
    <FirstName xmlns="">John</FirstName>
    <Surname xmlns="">Smith</Surname>
    <ExtraInformation xmlns="http://www.example.org/ExtraInformation">
    <!-- This 'ExtraInformation' tag is associated with the optional xsd:any
    from the XML schema.
    It can contain any well formed XML. -->
    <ExampleField1>one</ExampleField1>
    <ExampleField2>two</ExampleField2>
    </ExtraInformation>
    </Customer>
    </SOAP-ENV:Body>
    </SOAP-ENV:Envelope>
```

この SOAP メッセージが CICS に送信されると、CICS は Customer-xml-cont コンテナーに次に示す XML データを追加します。

```
<ExtraInformation
    xmlns="http://www.example.org/ExtraInformation">
    <!-- This 'ExtraInformation' tag is associated with the optional xsd:any
    from the XML schema.
    It can contain any well formed XML. -->
    <ExampleField1>one</ExampleField1>
    <ExampleField2>two</ExampleField2>
    </ExtraInformation>
```

CICS はまた Customer-xmlns-cont コンテナーに、次に示すスコープ内の XML 名前空間宣言を追加します。これらの宣言はスペースで区切られています。

```
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns="http://www.example.org/anyExample"
```

xsd:anyType の例

xsd:anyType とは、すべての単純および複合データ・タイプの派生元となる基本のデータ・タイプです。 これはデータ内容を制限しません。 データ・タイプを指定しないと、デフォルトとして xsd:anyType に設定されます。例えば、これらの 2 つの XML フラグメントは等価です。

```
<xsd:element name="Name" type="xsd:anyType"/>
```

```
<xsd:element name="Name"/>
```

生成される言語構造

< xsd:any > または xsd:anyType 用に生成される言語構造の形式は、COBOL では次のようになります。他の言語ではこれに相当する形式になります。

elementName-xml-cont PIC X(16)

未加工の XML を格納するコンテナーの名前。 CICS は、着信 SOAP メッセージを処理する際、<xsd:any> または xsd:anyType が定義する SOAP メッセージのサブセットをこのコンテナーに置きます。 アプリケーションは XML データをネイティブでのみ処理できます。アプリケーションは XML を生成し、このコンテナーにデータを取り込み、コンテナー名を提供する必要があります。

このコンテナーへの取り込みはテキスト・モードで行われる必要があります。 CICS がこのコンテナーにデータを取り込む場合、Web サービスが使用するように定義された EBCDIC のバリアントと同じものを使用して取り込みが行われます。ターゲットの EBCDIC コード・ページに存在しない文字は置換文字によって置き換えられます。アプリケーションによってコンテナーが UTF-8 で読み取られる場合も同様です。

elementName-xmlns-cont PIC X(16)

スコープ内の任意の名前空間の接頭部宣言を格納しているコンテナーの名前。 SOAP エンベロープ・タグのサブセットだけではなく、スコープ内にあり関連性のある名前空間の宣言がすべて含まれている点を除けば、このコンテナーの内容は DFHWS-XMLNS コンテナーの内容と似ています。

このコンテナーへの取り込みはテキスト・モードで行われる必要があります。 CICS がこのコンテナーにデータを取り込む場合、Web サービスが使用するように定義された EBCDIC のバリアントと同じものを使用して取り込みが行われます。ターゲットの EBCDIC コード・ページに存在しない文字は置換文字によって置き換えられます。アプリケーションによってコンテナーが UTF-8 で読み取られる場合も同様です。

このコンテナーは CICS に送信された SOAP メッセージを処理するときにのみ使用されます。出力 SOAP メッセージの生成時にアプリケーションがコンテナーに名前空間宣言を提供しようとする場合、コンテナーとその内容は CICS によって無視されます。 CICS では、アプリケーションが提供する XML に名前空間宣言が含まれていることが必要とされます。

<xsd:any> エレメントを含んでいる XML エレメントの名前は、<xsd:any> エレメント用に生成される変数名に組み込まれます。 <xsd:any> の例では、<xsd:any> エレメントは <xsd:element name="Customer"> エレメント内にネストされており、<xsd:any> エレメント用に生成される変数名は Customer-xml-cont PIC X(16) および Customer-xmlns-cont PIC X(16) です。

xsd:anyType タイプの場合、直接の XML エレメント名が使用されます。上記の xsd:anyType の例では、変数名は Name-xml-cont PIC X(16) および Name-xmlns-cont PIC X(16) です。

<xsd:choice> のサポート:

<xsd:choice> エレメントは、そのエレメントでオプションが 1 つだけ使用できることを示します。 CICS アシスタントは、さまざまなマッピング・レベルの <xsd:choice> エレメントに、さまざまな度合いのサポートを提供しています。

マッピング・レベル **2.2** 以上の **<xsd:choice>** のサポート

マッピング・レベル 2.2 以上では、DFHWS2LS および DFHSC2LS の <xsd:choice> エレメントのサポートが改善されています。 アシスタントは <xsd:choice> エレメントに関連した値を格納する新しいコンテナーを生成します。アシスタントは、以下に示すように新規コンテナー名および追加のフィールド名を含む言語構造を生成します。

***fieldname* -enum**
: <xsd:choice> エレメントが使用するオプションを区別して示すフィールド。

***fieldname* -cont**
: 使用されるオプションを格納するコンテナーの名前。 オプションの値をマップするために、さらに言語構造が生成されます。

次に示す XML スキーマのフラグメントには <xsd:choice> エレメントが含まれています。

```
<xsd:element name="choiceExample">
<xsd:complexType>
<xsd:choice>
<xsd:element name="option1" type="xsd:string" />
<xsd:element name="option2" type="xsd:int" />
<xsd:element name="option3" type="xsd:short" maxOccurs="2" minOccurs="2" />
</xsd:choice>
</xsd:complexType>
</xsd:element>
```

この XML スキーマ・フラグメントがマッピング・レベル 2.2 以上で処理される場合、アシスタントは次に示す COBOL 言語構造を生成します。

```
 03 choiceExample.
06 choiceExample-enum PIC X DISPLAY.
88 empty VALUE X'00'.
88 option1 VALUE X'01'.
88 option2 VALUE X'02'.
88 option3 VALUE X'03'.
06 choiceExample-cont PIC X(16).


01 Example-option1.
03 option1-length PIC S9999 COMP-5 SYNC.
03 option1 PIC X(255).

01 Example-option2.
03 option2 PIC S9(9) COMP-5 SYNC.

01 Example-option3.
03 option3 OCCURS 2 PIC S9999 COMP-5 SYNC.
```

マッピング・レベル **2.2** 以上における **<xsd:choice>** の制限

DFHSC2LS および DFHWS2LS はネストされた <xsd:choice> エレメントをサポートしません。例えば、次に示す XML はサポートされません。

```
<xsd:choice>
<xsd:element name ="name1" type="string"/>
<xsd:choice>
<xsd:element name ="name2a" type="string"/>
<xsd:element name ="name2b" type="string"/>
</xsd:choice>
</xsd:choice>
```

DFHSC2LS および DFHWS2LS は循環する <xsd:choice> エレメントをサポートしません。例えば、次の XML はサポートされません。

```
<xsd:choice maxOccurs="2">
<xsd:element name ="name1" type="string"/>
</xsd:choice>
```

DFHSC2LS および DFHWS2LS では、1 つの <xsd:choice> エレメントで最大 255 までのオプションをサポートします。

マッピング・レベル **2.1** 以下の **<xsd:choice>** のサポート

2.1 以下のマッピング・レベルでは、DFHWS2LS が <xsd:choice> エレメントに関して提供するサポートが限定されます。 DFHWS2LS は <xsd:choice> エレメントのそれぞれのオプションを、最大で 1 回生じる可能性のある <xsd:sequence> エレメントのように扱います。

<xsd:choice> エレメントのオプションのうち 1 つしか使用できないため、<xsd:choice> エレメントを使用してアプリケーションを実装する場合は有効なオプションの組み合わせのみが生成されるかどうかに注意してください。 それぞれのエレメントの生成された言語構造には count フィールドがありますが、そのうちの 1 つが 1 に、他のすべてが 0 に設定される必要があります。これ以外の組み合わせはすべて無効です。例外は、<xsd:choice> 自体がオプションの場合であり、すべてのフィールドが 0 に設定されても有効です。

置換グループのサポート:

置換グループを使用して、交換可能な XML エレメントのグループを定義できます。 CICS アシスタントはマッピング・レベル 2.2 以上で置換グループをサポートしています。

マッピング・レベル 2.2 以上では、DFHSC2LS および DFHWS2LS は <xsd:choice> エレメントに使用するマッピングに似たマッピングを使用して置換グループをサポートします。アシスタントは列挙フィールドと新規のコンテナー名を、その言語構造で生成します。

次に示す XML スキーマ・フラグメントには 2 つの subGroupParent エレメントの配列が含まれており、それぞれ replacementOption1 または replacementOption2 と置換可能です。

```
<xsd:element name="subGroupExample"
    >
    <xsd:complexType>
    <xsd:sequence>
    <xsd:element ref="subGroupParent" maxOccurs="2" minOccurs="2" />
```

```
</xsd:sequence>
</xsd:complexType>
</xsd:element>

<xsd:element name="subGroupParent" type="xsd:anySimpleType" />
<xsd:element name="replacementOption1" type="xsd:int"
substitutionGroup="subGroupParent" />
<xsd:element name="replacementOption2" type="xsd:short"
substitutionGroup="subGroupParent" />
```

この XML フラグメントをアシスタントで処理すると、次に示す COBOL 言語構造が生成されます。

```
03 subGroupExample.
06 subGroupParent OCCURS2.
09 subGroupExample-enum PIC X DISPLAY.
88 empty VALUE X '00'.
88 replacementOption1 VALUE X '01'.
88 replacementOption2 VALUE X '02'.
88 subGroupParent VALUE X '03'.
09 subGroupExample-cont PIC X (16).



01 Example-replacementOption1.
03 replacementOption1 PIC S9(9) COMP-5 SYNC.

01 Example-replacementOption2.
03 replacementOption2 PIC S9999 COMP-5 SYNC.

01 Example-subGroupParent.
03 subGroupParent-length PIC S9999 COMP-5 SYNC.
03 subGroupParent PIC X(255).
```

置換グループについて詳しくは、XML Schema Part 1: Structures Second Edition を参照してください。

抽象エレメントおよび抽象データ・タイプのサポート:

CICS アシスタントはマッピング・レベル 2.2 以上で抽象エレメントおよび抽象データ・タイプをサポートします。 CICS アシスタントは、置換グループと同様の方法で抽象エレメントおよび抽象データ・タイプをマップします。

マッピング・レベル **2.2** 以上における抽象エレメントのサポート

マッピング・レベル 2.2 以上では、DFHSC2LS および DFHWS2LS は、抽象エレメントが置換グループの有効なメンバーではないことを除くと、抽象エレメントを置換グループとほぼ同じ方法で扱います。 置換可能なエレメントがない場合、抽象エレメントは `<xsd:any>` エレメントとして扱われ、マッピング・レベル 2.1 における `<xsd:any>` エレメントと同じマッピングを使用します。

次に示す XML スキーマ・フラグメントは、抽象エレメントの代わりに使用できる 2 つのオプションを指定します。 抽象エレメント自体は有効なオプションではありません。

```
<xsd:element name="abstractElementExample" >
<xsd:complexType>
<xsd:sequence>
<xsd:element ref="abstractElementParent" maxOccurs="2" minOccurs="2" />
</xsd:sequence>
</xsd:complexType>
</xsd:element>
```

```
<xsd:element name="abstractElementParent" type="xsd:anySimpleType"
abstract="true" />
<xsd:element name="replacementOption1" type="xsd:int"
substitutionGroup="abstractElementParent" />
<xsd:element name="replacementOption2" type="xsd:short"
substitutionGroup="abstractElementParent" />
```

この XML フラグメントをアシスタントで処理すると、次に示す COBOL 言語構造が生成されます。

```
 03 abstractElementExample.
06 abstractElementParent OCCURS 2.
09 abstractElementExample-enum PIC X DISPLAY.
88 empty VALUE X '00'.
88 replacementOption1 VALUE X '01'.
88 replacementOption2 VALUE X '02'.
09 abstractElementExample-cont PIC X (16).


01 Example-replacementOption1.
03 replacementOption1 PIC S9(9) COMP-5 SYNC.

01 Example-replacementOption2.
03 replacementOption2 PIC S9999 COMP-5 SYNC.
```

抽象エレメントについて詳しくは、XML Schema Part 0: Primer Second Editionを参照してください。

マッピング・レベル **2.2** 以上における抽象データ・タイプのサポート

マッピング・レベル 2.2 以上では、DFHSC2LS および DFHWS2LS が抽象データ・タイプを置換グループとして扱います。 アシスタントは列挙フィールドと新規のコンテナー名を、その言語構造で生成します。

次に示す XML スキーマ・フラグメントは、抽象タイプの代わりに使用できる 2 つのオプションを指定します。

```
<xsd:element name="AbstractDataTypeExample"
type="abstractDataType" />

<xsd:complexType name="abstractDataType" abstract="true">
<xsd:simpleContent>
<xsd:extension base="xsd:string" />
</xsd:simpleContent>
</xsd:complexType>
<xsd:complexType name="option1">
<xsd:simpleContent>
<xsd:restriction base="abstractDataType">
<xsd:length value="5" />
</xsd:restriction>
</xsd:simpleContent>
</xsd:complexType>
<xsd:complexType name="option2">
<xsd:simpleContent>
<xsd:restriction base="abstractDataType">
<xsd:length value="10" />
</xsd:restriction>
</xsd:simpleContent>
</xsd:complexType>
```

この XML フラグメントをアシスタントで処理すると、次に示す COBOL 言語構造が生成されます。

```
 03 AbstractDataTypeExamp-enum PIC X DISPLAY.
88 empty VALUE X'00'.
88 option1 VALUE X'01'.
88 option2 VALUE X'02'.
03 AbstractDataTypeExamp-cont PIC X(16).
```

言語構造は別々のコピーブックに生成されます。 option1 用に生成される言語構造は、以下のコピーブックに生成されます。

```
 03 option1 PIC X(5).
```

option2 用に生成される言語構造は、以下の別のコピーブックに生成されます。

```
 03 option2 PIC X(10).
```

抽象データ・タイプについて詳しくは、XML Schema Part 0: Primer Second Editionを参照してください。

COBOL における可変の繰り返しコンテンツの処理方法**:**

COBOL では、データの各インスタンスを扱うポインター演算を使用して、可変繰り返しコンテンツを処理することはできません。他のプログラミング言語にはこの制限はありません。 この例では、Web サービス・アプリケーションのために COBOL で可変の繰り返しコンテンツを処理する方法を示します。

この方法は、**TRANSFORM** API コマンドを使用する XML のアプリケーション・データへの変換にも適用できます。 次の WSDL 文書の例は、繰り返し回数が可変である 8 文字のストリングからなるアプリケーション・データを含む Web サービスを表します。

```
<?xml version="1.0"?>
     <definitions name="ExampleWSDL"
     targetNamespace="http://www.example.org/variablyRepeatingData/"
     xmlns="http://schemas.xmlsoap.org/wsdl/"
     xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
     xmlns:tns="http://www.example.org/variablyRepeatingData/"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema">

     <types>
     <xsd:schema targetNamespace="http://www.example.org/variablyRepeatingData/">
     <xsd:element name="applicationData">
     <xsd:complexType>
     <xsd:sequence>
     <xsd:element name="component" minOccurs="1" maxOccurs="unbounded">
     <xsd:simpleType>
     <xsd:restriction base="xsd:string">
     <xsd:length value="8"/>
     </xsd:restriction>
     </xsd:simpleType>
     </xsd:element>
     </xsd:sequence>
     </xsd:complexType>
     </xsd:element>
     </xsd:schema>
     </types>

     <message name="exampleMessage">
     <part element="tns:applicationData" name="messagePart"/>
     </message>

     <portType name="examplePortType">
     <operation name="exampleOperation">
     <input message="tns:exampleMessage"/>
     <output message="tns:exampleMessage"/>
     </operation>
```

```
    </portType>

    <binding name="exampleBinding" type="tns:examplePortType">
    <soap:binding style="document"
    transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="exampleOperation">
    <soap:operation soapAction=""/>
    <input><soap:body parts="messagePart" encodingStyle="" use="literal"/></input>
    <output><soap:body parts="messagePart" encodingStyle=""
    use="literal"/></output>
    </operation>
    </binding>
    </definitions>
```

この WSDL 文書を DFHWS2LS を使用して処理すると、次に示す COBOL 言語構造が生成されます。

```
 03 applicationData.

06 component-num PIC S9(9) COMP-5 SYNC.
06 component-cont PIC X(16).


01 DFHWS-component.
03 component PIC X(8).
```

8 文字の component フィールドが、DFHWS-component という名前の別の構造で定義されていることに注意してください。メインのデータ構造は applicationData と呼ばれており、component-num および component-cont という 2 つのフィールドを含んでいます。 component-num フィールドは、component データのインスタンス数を示し、component-cont フィールドは component フィールドの連結リストを格納するコンテナーの名前を示します。

次に示す COBOL コードは、可変の繰り返しデータのリストを処理する方法の 1 つを例示しています。ここでは linkage section 配列を使用してデータの後続インスタンスを処理しています。インスタンスはそれぞれ DISPLAY ステートメントを使用して表示されます。

```
IDENTIFICATION DIVISION.
PROGRAM-ID. EXVARY.

ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.

* working storage variables
01 APP-DATA-PTR USAGE IS POINTER.
01 APP-DATA-LENGTH PIC S9(8) COMP.
01 COMPONENT-PTR USAGE IS POINTER.
01 COMPONENT-DATA-LENGTH PIC S9(8) COMP.
01 COMPONENT-COUNT PIC S9(8) COMP-4 VALUE 0.
01 COMPONENT-LENGTH PIC S9(8) COMP.

LINKAGE SECTION.

* a large linkage section array
01 BIG-ARRAY PIC X(659999).

* application data structures produced by DFHWS2LS
* this is normally referenced with a COPY statement
01 DFHWS2LS-data.
03 applicationData.
06 component-num PIC S9(9) COMP-5 SYNC.
06 component-cont PIC X(16).
```

```
01 DFHWS-component.
03 component PIC X(8).



PROCEDURE DIVISION USING DFHEIBLK.
A-CONTROL SECTION.
A010-CONTROL.

* Get the DFHWS-DATA container
EXEC CICS GET CONTAINER('DFHWS-DATA')
SET(APP-DATA-PTR)
FLENGTH(APP-DATA-LENGTH)
END-EXEC
SET ADDRESS OF DFHWS2LS-data TO APP-DATA-PTR

* Get the recurring component data
EXEC CICS GET CONTAINER(component-cont)
SET(COMPONENT-PTR)
FLENGTH(COMPONENT-DATA-LENGTH)
END-EXEC

* Point the component structure at the first instance of the data
SET ADDRESS OF DFHWS-component TO COMPONENT-PTR

* Store the length of a single component
MOVE LENGTH OF DFHWS-component TO COMPONENT-LENGTH

* process each instance of component data in turn
PERFORM WITH TEST AFTER
UNTIL COMPONENT-COUNT = component-num

* display the current instance of the data
DISPLAY 'component value is: ' component

* address the next instance of the component data
SET ADDRESS OF BIG-ARRAY TO ADDRESS OF DFHWS-component
SET ADDRESS OF DFHWS-component
TO ADDRESS OF BIG-ARRAY (COMPONENT-LENGTH + 1:1)
ADD 1 TO COMPONENT-COUNT

* end the loop
END-PERFORM.

* Point the component structure back at the first instance of
* of the data, for any further processing we may want to perform
SET ADDRESS OF DFHWS-component TO COMPONENT-PTR

* return to CICS.

EXEC CICS
RETURN
END-EXEC

GOBACK.
```

上記のコードは可変の繰り返しコンテンツの一般的な処理方法を提供しています。配列 BIG-ARRAY はそれぞれのコンポーネントの先頭に順次移動し、データの先頭に固定されません。 コンポーネントのデータ構造は、次のコンポーネントの最初のバイトを指すように移動します。 COMPONENT-PTR を使用して、必要に応じてコンポーネント・データの開始位置を回復できます。

WSDL 文書に準拠する SOAP メッセージの例を次に示します。

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Body>
<applicationData xmlns="http://www.example.org/variablyRepeatingData/">
<component xmlns="">VALUE1</component>
<component xmlns="">VALUE2</component>
<component xmlns="">VALUE3</component>
</applicationData>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

COBOL プログラムが SOAP メッセージを処理する際に生成される出力は次のとおりです。

```
CPIH 20080115103151 component value is: VALUE1
CPIH 20080115103151 component value is: VALUE2
CPIH 20080115103151 component value is: VALUE3
```

変化するエレメントの配列

XML は、エレメント数が変化する配列を持つことができます。一般に、エレメント数が変化する WSDL 文書や XML スキーマは 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。 CICS はコンテナー・ベースのマッピングまたはインライン・マッピングを使用して XML におけるエレメント数の変化に対応します。

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。

- 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。

これは、アプリケーションの XML または 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。

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

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:length value="8"/>
</xsd:restriction>
</xsd:simpleType>
</xsd:element>
```

一般に、エレメント数が変化する WSDL 文書は 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 は一連のコンテナーとしてアプリケーション・プログラム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。これらの構造はアプリケーションへの入出力として使用されます。

- CICS が XML をアプリケーション・データに変換する場合、これらの構造にアプリケーション・データが追加され、アプリケーションがそのデータを読み取ります。
- CICS がアプリケーション・データを XML に変換する場合、アプリケーションによって構造内に追加されたアプリケーション・データが読み取られます。

これらのデータ構造のフォーマットは、一連の例を使用すると分かりやすくなります。 SOAP メッセージまたはアプリケーションの XML を使用できます。 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:length 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
```

マッピング・レベル 2 以下における変化するエレメント数

この例では、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:length value="8"/>
</xsd:restriction>
</xsd:simpleType>
</xsd:element>
```

主データ構造には、2 つのフィールドの宣言が格納されます。 CICS が XML をバイナリー・データに変換すると、最初のフィールドである component-num にはエレメントが XML 内に現れる回数が格納され、2 番目のフィールドである component-cont には、コンテナーの名前が格納されます。

```
05 component-num PIC S9(9) COMP-5
05 component-cont PIC X(16)
```

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)
```

エレメントが発生する回数を割り出すには、component-num (1 から 5 までの範囲の値が入ります) の値を検査する必要があります。 エレメント・コンテンツは component-cont で指定されたコンテナー内にあります。このコンテナーはエレメントの配列を保持しており、各エレメントは DFHWS-component データ構造によってマップされています。

minOccurs="0" および maxOccurs="1" である場合、このエレメントはオプションです。 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、以下のように component-num 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 エレメントはなく、component-cont の内容は未定義です。
- この値が 1 の場合、component エレメントは component-cont 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。

注: SOAP メッセージが単一の繰り返しエレメントで構成される場合、DFHWS2LS は 2 つの言語構造を生成します。 主要な言語構造は、配列エレメントの数と、エレメントの配列を保持するコンテナーの名前を含んでいます。 2 番目の言語構造は、繰り返しエレメントの単一インスタンスをマップします。

マッピング・レベル 2.1 以上における変化するエレメント数

マッピング・レベル 2.1 以上では、CICS アシスタントで **INLINE-MAXOCCURS-LIMIT** パラメーターを使用できます。 **INLINE-MAXOCCURS-LIMIT** パラメーターは変化するエレメント数を処理する方法を指定します。 可変数のエレメント用のマッピング・オプションは、コンテナー・ベースのマッピング (390 ページの『マッピング・レベル 2 以下における変化するエレメント数』で説明されています) またはインライン・マッピングです。 このパラメーターの *value* は、0 - 32767 までの範囲の正整数です。

- **INLINE-MAXOCCURS-LIMIT** のデフォルト値は 1 であり、オプション・エレメントがインラインで確実にマップされます。
- **INLINE-MAXOCCURS-LIMIT** パラメーターに値 0 が設定されている場合、インライン・マッピングは使用されません。
- maxOccurs が **INLINE-MAXOCCURS-LIMIT** の値以下の場合、インライン・マッピングが使用されます。
- maxOccurs が **INLINE-MAXOCCURS-LIMIT** の値以上の場合、コンテナー・ベースのマッピングが使用されます。

変化するエレメント数をインラインでマップすると、配列 (上記の例では発生するオカレンスが固定) およびカウンターが生成されます。 component-num フィールドは存在するエレメントのインスタンス数を示し、それらのインスタンスは配列によって示されます。 390 ページの『マッピング・レベル 2 以下における変化するエレメント数』で示される例では、**INLINE-MAXOCCURS-LIMIT** が 5 以下の場合に生成されるデータ構造は次のようになります。

```
05 component-num PIC S9(9) COMP-5 SYNC.
05 component OCCURS 5 PIC X(8).
```

最初のフィールド (component-num) は前のセクションで示されたコンテナー・ベース・マッピングの例の出力と同じです。 2 番目のフィールドには長さが 5 の配列があり、生成される可能性のある最大エレメント数を収容できる大きさです。

インライン・マッピングは、エレメントの出現回数およびデータが収容されるコンテナーの名前を格納するコンテナー・ベースのマッピングとは異なり、現行のコンテナーにあるすべてのデータを格納します。 現行のコンテナーにデータを格納するとパフォーマンスが向上するので、インライン・マッピングの方が一般的には望ましいといえます。

ネストされた変数配列

複雑な WSDL 文書および XML スキーマには可変の繰り返しエレメントを含めることができ、そのエレメントにはさらに可変の繰り返しエレメントを含めることができます。 この場合、記述される構造は、例で説明した 2 つのレベルを超えます。

この例は、1 回から 5 回出現する可能性がある必須エレメント (<component1>) 内でネストされたオプションのエレメント (<component2>) を示します。

```
<xsd:element name="component1"
minOccurs="1" maxOccurs="5">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="component2"
minOccurs="0" maxOccurs="1">
<xsd:simpleType>
<xsd:restriction base="xsd:string">
<xsd:length value="8"/>
</xsd:restriction>
</xsd:simpleType>
</xsd:element>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
```

最上位のデータ構造は、前の例のデータ構造とまったく同じです。

```
05 component1-num PIC S9(9) COMP-5
05 component1-cont PIC X(16)
```

ただし、2 番目のデータ構造には、以下のエレメントが格納されます。

```
01 DFHWS-component1
02 component2-num PIC S9(9) COMP-5
02 component2-cont PIC X(16)
```

3 番目の構造には以下のエレメントが格納されます。

```
01 DFHWS-component2
02 component2 PIC X(8)
```

最外部エレメント <component1> の出現回数は、component1-num に格納されます。

component1-cont で指定されたコンテナーには、2 番目のデータ構造 (DFHWS-component1) のその数のインスタンスを持つ配列が含まれています。

component2-cont の各インスタンスは、異なるコンテナーを指定します。それぞれ、第 3 レベルの構造 (DFHWS-component2) によってマップされたデータ構造が含まれています。

この構造を説明するために、例と一致する次のような XML のフラグメントについて考えてみます。

```
<component1><component2>
string1
</component2></component1>
<component1><component2>
string2
</component2></component1>
<component1></component1>
```

<component1> は 3 回発生します。 最初の 2 つには、それぞれ <component2> のインスタンスが含まれていますが、3 番目のインスタンスには含まれていません。

最上位のデータ構造では、component1-num に値 3 が含まれています。component1-cont で指定されたコンテナーには、DFHWS-component1 の次の 3 つのインスタンスがあります。

1. 第 1 のインスタンスでは、component2-num の値は 1 で、component2-cont で指定されたコンテナーは *string1* を保持します。
2. 第 2 のインスタンスでは、component2-num の値は 1 で、component2-cont で指定されたコンテナーは *string2* を保持します。
3. 第 3 のインスタンスでは、component2-num の値は 0 で、component2-cont の内容は未定義です。

このインスタンスでは、完全なデータ構造は、次の合計 4 つのコンテナーによって表現されます。

- コンテナー DFHWS-DATA 内のルート・データ構造
- component1-cont で指定されたコンテナー
- component2-cont の最初の 2 つのインスタンスで指定された 2 つのコンテナー

オプションの構造および xsd:choice

DFHWS2LS および DFHSC2LS はマッピング・レベル 2.1 以上で、<xsd:sequence>、<xsd:choice>、および <xsd:all> エレメントでの maxOccurs および minOccurs の使用をサポートします。この場合、minOccurs 属性および maxOccurs 属性は minOccurs="0" および maxOccurs="1" に設定されています。

アシスタントは、エレメント内の子エレメントがそれぞれオプションであるかのようにしてエレメントを処理するマッピングを生成します。 これらのエレメントを使用してアプリケーションを実装する場合、アプリケーションが無効なオプションの組み合わせを生成しないように確認してください。 それぞれのエレメントの生成された言語構造には count フィールドがありますが、これらのすべてが「0」に設定されるか、すべてが「1」に設定される必要があります。これ以外の値の組み合わせは、<xsd:choice> エレメントの場合を除き、無効です。

<xsd:choice> エレメントは、そのエレメントのオプションが 1 つのみ使用できることを示します。 これはすべてのマッピング・レベルでサポートされます。 アシス

タントは <xsd:choice> のそれぞれのオプションを、minOccurs="0" および maxOccurs="1" である <xsd:sequence> エレメントのように扱います。 <xsd:choice> エレメントを使用してアプリケーションを実装する場合は、アプリケーションが無効なオプションの組み合わせを生成しないように確認してください。 それぞれのエレメントの生成された言語構造には count フィールドがありますが、そのうちの 1 つが「1」に、他のすべてが「0」に設定される必要があります。 これ以外の組み合わせはすべて無効です。例外は、<xsd:choice> のエレメント自体がオプションの場合であり、すべてのフィールドが 0 に設定されていても有効です。

可変長の値と空白のサポート

CICS 支援機能の設定値を使用したり、XML スキーマに直接ファセットを追加したりすることにより、可変長の値と空白を処理する方法をカスタマイズできます。

通常、CICS XML 支援機能と CICS Web サービス支援機能は、データ・ストリングを固定長の文字配列にマップします。こうした配列では、スペースや NULL で埋め込む必要があります。 可変長の値を固定長のデータ配列にマッピングすると、効率が悪く、ストレージを浪費する恐れがあります。使用するデータ長が可変の場合、こうしたマッピングを処理する方法をカスタマイズすることが推奨されています。

ある言語構造から XML スキーマまたは WSDL 文書に変換する場合には、XML スキーマで whiteSpace ファセットと maxLength ファセットを指定し、支援機能で **CHAR-VARYING-LIMIT** パラメーターを設定することが推奨されています。

XML スキーマまたは WSDL 文書からある言語構造に変換する場合には、支援機能で **CHAR-VARYING** パラメーターを適切な値に設定することが推奨されています。

注: XML 文書では、NULL 文字 ('x00') は無効です。 CICS が構文解析するアプリケーション・データの NULL 文字はストリングの末尾を示すためのもので、値は切り捨てられます。 CICS は、アプリケーション・データを生成する際に、**CHAR-VARYING** パラメーターの値に従って生成します。 例えば、**CHAR-VARYING=NULL** オプションが指定されている場合、CICS によって生成される可変長のストリングはヌル文字で終わります。

XML から言語構造への可変長の値のマッピング

XML スキーマのファセットを使用して、または CICS 支援機能で特定のパラメーターを指定して、XML スキーマまたは WSDL 文書と言語構造間のマッピングを処理する方法をカスタマイズします。

ファセットを使用して XML データ・タイプを制限することができます。 XML 内の可変長データの処理方法をカスタマイズするには、長さファセット (length、maxLength、および minLength) および whiteSpace ファセットを使用します。

length
: データが固定長となるように指定します。

maxLength
: データ・タイプの最大長を指定します。 ストリング・ベースのデータ・タイプでこの値を設定しないと、最大長は無制限になります。

minLength
データ・タイプの最小長を指定します。 ストリング・ベースのデータ・タイプでこの値を設定しないと、最小長は 0 になります。

whiteSpace
データ値の周囲にある空白の処理方法を指定します。空白には、スペース、タブ、および改行が含まれます。 whiteSpace ファセットは、preserve、replace、または collapse に設定できます。

- 値 preserve は、データ値内の空白をすべて保持します。
- 値 replace は、タブまたは改行が適切な数のスペースに置換されることを意味します。
- 値 collapse は、先行空白、後続空白、および埋め込み空白は除去され、タブ、改行、および連続スペースはすべて単一のスペース文字に置き換えられることを意味します。

whiteSpace ファセットが設定されないと、空白は保持されます。

XML スキーマ・ファセットについて詳しくは、XML Schema Part 2: Datatypes Second Editionを参照してください。

CICS 支援機能でパラメーター DFHSC2LS および DFHWS2LS を使用すると、XML スキーマから言語構造への可変長データのマッピング方法を変更できます。これらのパラメーターは、マッピング・レベル 1.2 以降で使用可能です。

DEFAULT-CHAR-MAXLENGTH
XML スキーマ文書または WSDL 文書で長さが指定されていない場合に、マッピングする文字の文字データのデフォルトの配列長を指定します。 このパラメーターの値は、1 から 2 147 483 647 の範囲の正整数です。

ただし、XML スキーマまたは WSDL 文書で maxLength ファセットを使用して、DFHSC2LS または DFHWS2LS が使用する最大文字長を直接指定することをお勧めします。 XML スキーマまたは WSDL 文書で最大長を直接指定すると、すべてのストリング・ベースのデータ・タイプに 1 つのグローバル・デフォルトが適用されることに関連した問題を回避できます。

CHAR-VARYING-LIMIT
言語構造にマップされる可変長文字データの最大サイズを指定します。 文字データが、このパラメーターで指定されている値よりも大きい場合は、コンテナーにマップされ、生成された言語構造でそのコンテナー名が使用されます。値は 0 からデフォルトの 32 767 バイトの範囲で指定できます。

CHAR-VARYING
可変長文字データがどのようにマップされるのかを指定します。 このパラメーターを指定しない場合は、指定される言語に応じてデフォルトのマッピングが異なります。 以下のオプションを選択できます。

- **CHAR-VARYING=NO** は、可変長文字データが固定長ストリングとしてマップされるように指定します。
- **CHAR-VARYING=NULL** は、可変長文字データがヌル終了ストリングにマップされるように指定します。
- **CHAR-VARYING=YES** は、可変長文字データが PL/I で CHAR VARYING データ・タイプにマップされるように指定します。

COBOL、C、および C++ 言語の場合、可変長文字データは、2 つの関連エレメント (データ長およびデータ) で構成される同等の表現にマップされます。

通常、**CHAR-VARYING=YES** に設定すると、最高のパフォーマンスが得られます。

言語構造から XML への可変長の値のマッピング

ご使用の言語構造と XML スキーマまたは WSDL 文書間におけるマッピングの処理方法をカスタマイズできます。 DFHLS2SC または DFHLS2WS の **CHAR-VARYING** パラメーターを COLLAPSE または NULL に設定して、文字配列が生成される方法を変更します。

CHAR-VARYING=NULL オプションを設定すると、CICS に XML を生成する際に各文字配列の末尾に NULL 文字を追加するように指示することになります。

CHAR-VARYING=COLLAPSE オプションを設定すると、CICS に XML を生成する際に文字配列の末尾から後続スペースを自動的に除去するように指示することになります。 このオプションが有効なのはマッピング・レベルが 2.1 以上の場合だけで、C および C++ 以外のすべての言語ではマッピング・レベル 2.1 以降のデフォルト値が **CHAR-VARYING=COLLAPSE** になります。 XML が解析されると、先行空白、後続空白、埋め込み空白はすべて削除されます。

DFHJS2LS 内のエレメントの可変配列

JSON には、エレメント数が変化する配列を含めることができます。一般に、エレメント数が変化する JSON スキーマは 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。 CICS はコンテナー・ベースのマッピングまたはインライン・マッピングを使用して JSON データにおけるエレメント数の変化に対応します。

エレメント数が変化する配列を JSON スキーマ内で表現するには、"type" 値が "array" のスキーマで、キーワード minItems と maxItems を使用します。

- minItems キーワードは、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。デフォルトは、値 0 です。
- maxItems キーワードは、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、minItems キーワードの値以上の任意の正の整数値を指定できます。
- maxItems キーワードがない場合、配列が無制限であることを示します。

可変配列 "maxItems":1 によってオプション・フィールドを表すことができます。例えば、"component" という名前の 8 バイトのオプション・ストリングは次のようになります。

```
"properties":{
"component": {
"type":"array",
"maxItems":1,
"items": {
"type": "string",
"maxLength": 8
```

```
}
}
},
"required": ["component"]
```

次のように、"required" キーワード値にフィールド名を含めないことにより、同じ効果を得ることができます。

```
"properties":{
"component": {
"type": "string",
"maxLength": 8
}
}
```

一般に、エレメント数が変化する JSON スキーマは 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。 このような場合に対処するため、CICS は一連のコンテナーとしてアプリケーション・プログラム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。 これらの構造はアプリケーションへの入出力として使用されます。

- CICS が JSON データをアプリケーション・データに変換する場合、これらの構造にアプリケーション・データが追加され、アプリケーションがそのデータを読み取ります。
- CICS がアプリケーション・データを JSON データに変換する場合、アプリケーションによって構造内に追加されたアプリケーション・データが読み取られます。

以下の例は、こうしたデータ構造のフォーマットを示しています。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。

例 1. 固定のエレメント数

この例では、ちょうど 3 回発生するエレメントを示しています。

```
"properties":{
"component": {
"type": "array",
"maxItems": 3,
"minItems": 3,
"items": {
"type": "string",
"maxLength": 8
}
}
},
"required": ["component"]
```

この例では、エレメントが出現回数が事前に分かっているので、単純な COBOL 宣言、または他の言語のこれに相当するもので固定長配列として表すことができます。

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

例 2. マッピング・レベル 2 以下における変化するエレメント数

この例では、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。

```
"properties":{
"component": {
"type": "array",
"maxItems": 5,
"minItems": 1,
"items": {
"type": "string",
"maxLength": 8
}
}
},
"required": ["component"]
```

主データ構造には、2 つのフィールドの宣言が格納されます。 CICS が JSON データをバイナリー・データに変換すると、最初のフィールドである component-num にはエレメントがJSON データ内に現れる回数が格納され、2 番目のフィールドである component-cont には、コンテナーの名前が格納されます。

```
05 component-num PIC S9(9) COMP-5
05 component-cont PIC X(16)
```

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 DFHJS-component
02 component PIC X(8)
```

エレメントの出現回数を確認するには、component-num (1 から 5 の範囲の値が含まれる) の値を調べる必要があります。エレメント・コンテンツは component-cont で指定されたコンテナー内にあります。このコンテナーはエレメントの配列を保持しており、各エレメントは DFHJS-component データ構造によってマップされています。

minItems="0" (または未指定) で、かつ maxItems="1" の場合、このエレメントはオプションです。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、以下のように component-num 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 エレメントはなく、component-cont の内容は未定義です。
- この値が 1 の場合、component エレメントは component-cont 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、DFHJS-component データ構造によってマップされます。

注**:** JSON データが単一の繰り返しエレメントで構成される場合、DFHJS2LS は 2 つの言語構造を生成します。 主要な言語構造は、配列エレメントの数と、エレメントの配列を保持するコンテナーの名前を含んでいます。 2 番目の言語構造は、繰り返しエレメントの単一インスタンスをマップします。

例 3 マッピング・レベル 2.1 以上における変化するエレメント数

マッピング・レベル 2.1 では、CICS 支援機能で **INLINE-MAXOCCURS-LIMIT** パラメーターを使用できます。 **INLINE-MAXOCCURS-LIMIT** パラメーターは変化するエレメント数を処理する方法を指定します。 変化するエレメント数のマッピング・オプションは、 325 ページの『例 2. マッピング・レベル 2 以下における変化するエレメント数』に記述されているコンテナー・ベースのマッピング、またはインライン・マッピングです。 このパラメーターの値は、0 - 32767 までの範囲の正整数です。

- **INLINE-MAXOCCURS-LIMIT** のデフォルト値は 1 であり、オプション・エレメントがインラインで確実にマップされます。
- **INLINE-MAXOCCURS-LIMIT** パラメーターに値 0 が設定されている場合、インライン・マッピングは使用されません。
- maxItems が **INLINE-MAXOCCURS-LIMIT** の値以下の場合、インライン・マッピングが使用されます。
- maxItems が **INLINE-MAXOCCURS-LIMIT** の値以上の場合、コンテナー・ベースのマッピングが使用されます。

変化するエレメント数をインラインでマップすると、配列 (上記の例では発生するオカレンスが固定) およびカウンターが生成されます。 component-num フィールドは存在するエレメントのインスタンス数を示し、それらのインスタンスは配列によって示されます。 325 ページの『例 2. マッピング・レベル 2 以下における変化するエレメント数』で示される例では、**INLINE-MAXOCCURS-LIMIT** が 5 以下の場合に生成されるデータ構造は次のようになります。

```
05 component-num PIC S9(9) COMP-5 SYNC.
05 component OCCURS 5 PIC X(8).
```

最初のフィールド (component-num) は前のセクションで示されたコンテナー・ベース・マッピングの例の出力と同じです。 2 番目のフィールドには長さが 5 の配列があり、生成される可能性のある最大エレメント数を収容できる大きさです。

インライン・マッピングは、エレメントの出現回数およびデータが収容されるコンテナーの名前を格納するコンテナー・ベースのマッピングとは異なり、現行のコンテナーにあるすべてのデータを格納します。 現行のコンテナーにデータを格納するとパフォーマンスが向上するので、インライン・マッピングの方が一般的には望ましいといえます。

例 4. ネストされた変数配列

複雑な JSON スキーマには可変の繰り返しエレメントを含めることができ、そのエレメントにはさらに可変の繰り返しエレメントを含めることが可能です。この場合、記述される構造は、例で説明した 2 つのレベルを超えます。

この例は、1 回から 5 回出現する可能性がある必須エレメント "component1" 内でネストされたオプションのエレメント "component2" を示しています。

```
"properties":{
"component1": {
"type": "array",
"maxItems": 5,
"minItems": 1,
"items": {
"type": "object",
"properties":{
"component2":{
"type": "string",
"maxLength": 8
}
},
"required": ["component2"]
}
},
"required": ["component1"]
```

最上位のデータ構造は、前の例のデータ構造とまったく同じです。

```
05 component1-num PIC S9(9) COMP-5
05 component1-cont PIC X(16)
```

ただし、2 番目のデータ構造には、以下のエレメントが格納されます。

```
01 DFHJS-component1
02 component2-num PIC S9(9) COMP-5
02 component2-cont PIC X(16)
```

3 番目の構造には以下のエレメントが格納されます。

```
01 DFHJS-component2
02 component2 PIC X(8)
```

最外部エレメント "component1" の出現回数は、component1-num に格納されます。

component1-cont で指定されたコンテナーには、2 番目のデータ構造 (DFHJS-component1) のその数のインスタンスを持つ配列が含まれています。

component2-cont の各インスタンスは、異なるコンテナーを指定します。それぞれ、第 3 レベルの構造 (DFHJS-component2) によってマップされたデータ構造が含まれています。

この構造を説明するために、例と一致する次のような JSON データのフラグメントについて考えてみます。

```
{"component1":
[
{
"component2": "string1"
},
{
"component2": "string2"
},
]
}
```

"component1" は 3 回発生します。 最初の 2 つには、"component2" のインスタンスが含まれていますが、3 番目のインスタンスには含まれていません。

最上位のデータ構造では、component1-num に値 3 が含まれています。component1-cont で指定されたコンテナーには、DFHJS-component1 の次の 3 つのインスタンスがあります。

1. 第 1 のインスタンスでは、component2-num の値は 1 で、component2-cont で指定されたコンテナーは *string1* を保持します。
2. 第 2 のインスタンスでは、component2-num の値は 1 で、component2-cont で指定されたコンテナーは *string2* を保持します。
3. 第 3 のインスタンスでは、component2-num の値は 0 で、component2-cont の内容は未定義です。

このインスタンスでは、完全なデータ構造は、次の合計 4 つのコンテナーによって表現されます。

- コンテナー DFHJS-DATA 内のルート・データ構造。
- component1-cont で指定されたコンテナー。

- component2-cont の最初の 2 つのインスタンスで指定された 2 つのコンテナー。

オプションの構造と **required** キーワード

データ構造は、JSON スキーマ "type" の "object" で定義します。"properties" キーワードで指定したオブジェクトを使用して、スキーマによりフィールド名が個々のタイプと関連付けられます。これらのフィールドが、JSON スキーマによって記述された JSON データの一部となることが必要かどうかは、"required" キーワードで指定される配列により制御されます。この配列には、JSON データ内に存在する必要があるすべてのフィールド名がリストされます。したがって、オプションのフィールドを表すには、そのフィールドをこの配列から除きます。あるいは、配列を空にはできないので、"required" キーワードを除きます。後者の場合、すべてのフィールドがオプションとなります。

オプションのフィールドは、0 個または 1 個のエレメントから成る変数配列として扱われます。これは、エレメント名に接尾部 "-num" が付加されたフィールドを追加します。全長が 28 文字を超える場合、エレメント名が切り捨てられます。これは実行時に、JSON データに値が存在していたことを示す場合は非ゼロ、そうでない場合はゼロになります。

以下の例に 2 つのフィールドを示します。一方は "required-structure" という必須フィールドで、もう一方は "optional-structure" というオプション・フィールドです。

```
{
"type": "object",
"properties": {
"required-structure": {
"type":"string",
"maxLength": 8
},
"optional-structure": {
"type":"string",
"maxLength": 8
}
},
"required": [
"required-structure"
]
}
```

生成される COBOL 構造には両方のフィールドが示されますが、2 番目の前にある "optional-structure-num" はエレメントに関する整数カウントで、0 は「なし」、1 は存在することを表しています。この値は、"optional-structure" に有効なデータが含まれるかどうかを示すように設定されます。

```
03 OutputData.
06 required-structure PIC X(8).
06 optional-structure-num PIC S9(9) COMP-5 SYNC.
06 optional-structure PIC X(8).
```

変化するエレメントの配列

XML は、エレメント数が変化する配列を持つことができます。一般に、エレメント数が変化する WSDL 文書や XML スキーマは 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。 CICS はコンテナー・ベースのマッピングまたはインライン・マッピングを使用して XML におけるエレメント数の変化に対応します。

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。

- 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。

これは、アプリケーションの XML または 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。

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

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:length value="8"/>
</xsd:restriction>
</xsd:simpleType>
</xsd:element>
```

一般に、エレメント数が変化する WSDL 文書は 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 は一連のコンテナーとしてアプリケーション・プログラム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。これらの構造はアプリケーションへの入出力として使用されます。

- CICS が XML をアプリケーション・データに変換する場合、これらの構造にアプリケーション・データが追加され、アプリケーションがそのデータを読み取ります。
- CICS がアプリケーション・データを XML に変換する場合、アプリケーションによって構造内に追加されたアプリケーション・データが読み取られます。

これらのデータ構造のフォーマットは、一連の例を使用すると分かりやすくなります。 SOAP メッセージまたはアプリケーションの XML を使用できます。 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:length 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
```

マッピング・レベル 2 以下における変化するエレメント数

この例では、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:length value="8"/>
</xsd:restriction>
</xsd:simpleType>
</xsd:element>
```

主データ構造には、2 つのフィールドの宣言が格納されます。 CICS が XML をバイナリー・データに変換すると、最初のフィールドである component-num にはエレメントが XML 内に現れる回数が格納され、2 番目のフィールドである component-cont には、コンテナーの名前が格納されます。

```
05 component-num PIC S9(9) COMP-5
05 component-cont PIC X(16)
```

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)
```

エレメントが発生する回数を割り出すには、component-num (1 から 5 までの範囲の値が入ります) の値を検査する必要があります。 エレメント・コンテンツは component-cont で指定されたコンテナー内にあります。このコンテナーはエレメントの配列を保持しており、各エレメントは DFHWS-component データ構造によってマップされています。

minOccurs="0" および maxOccurs="1" である場合、このエレメントはオプションです。 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、以下のように component-num 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 エレメントはなく、component-cont の内容は未定義です。
- この値が 1 の場合、component エレメントは component-cont 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。

注: SOAP メッセージが単一の繰り返しエレメントで構成される場合、DFHWS2LS は 2 つの言語構造を生成します。 主要な言語構造は、配列エレメントの数と、エレメントの配列を保持するコンテナーの名前を含んでいます。 2 番目の言語構造は、繰り返しエレメントの単一インスタンスをマップします。

マッピング・レベル 2.1 以上における変化するエレメント数

マッピング・レベル 2.1 以上では、CICS アシスタントで **INLINE-MAXOCCURS-LIMIT** パラメーターを使用できます。 **INLINE-MAXOCCURS-LIMIT** パラメーターは変化するエレメント数を処理する方法を指定します。 可変数のエレメント用のマッピング・オプションは、コンテナー・ベースのマッピング (390 ページの『マッピング・レベル 2 以下における変化するエレメント数』で説明されています) またはインライン・マッピングです。 このパラメーターの *value* は、0 - 32767 までの範囲の正整数です。

- **INLINE-MAXOCCURS-LIMIT** のデフォルト値は 1 であり、オプション・エレメントがインラインで確実にマップされます。
- **INLINE-MAXOCCURS-LIMIT** パラメーターに値 0 が設定されている場合、インライン・マッピングは使用されません。
- maxOccurs が **INLINE-MAXOCCURS-LIMIT** の値以下の場合、インライン・マッピングが使用されます。
- maxOccurs が **INLINE-MAXOCCURS-LIMIT** の値以上の場合、コンテナー・ベースのマッピングが使用されます。

変化するエレメント数をインラインでマップすると、配列 (上記の例では発生するオカレンスが固定) およびカウンターが生成されます。 component-num フィールドは存在するエレメントのインスタンス数を示し、それらのインスタンスは配列によって示されます。 390 ページの『マッピング・レベル 2 以下における変化するエレメント数』で示される例では、**INLINE-MAXOCCURS-LIMIT** が 5 以下の場合に生成されるデータ構造は次のようになります。

```
05 component-num PIC S9(9) COMP-5 SYNC.
05 component OCCURS 5 PIC X(8).
```

最初のフィールド (component-num) は前のセクションで示されたコンテナー・ベース・マッピングの例の出力と同じです。 2 番目のフィールドには長さが 5 の配列があり、生成される可能性のある最大エレメント数を収容できる大きさです。

インライン・マッピングは、エレメントの出現回数およびデータが収容されるコンテナーの名前を格納するコンテナー・ベースのマッピングとは異なり、現行のコンテナーにあるすべてのデータを格納します。 現行のコンテナーにデータを格納するとパフォーマンスが向上するので、インライン・マッピングの方が一般的には望ましいといえます。

ネストされた変数配列

複雑な WSDL 文書および XML スキーマには可変の繰り返しエレメントを含めることができ、そのエレメントにはさらに可変の繰り返しエレメントを含めることができます。 この場合、記述される構造は、例で説明した 2 つのレベルを超えます。

この例は、1 回から 5 回出現する可能性がある必須エレメント (<component1>) 内でネストされたオプションのエレメント (<component2>) を示します。

```
<xsd:element name="component1"
minOccurs="1" maxOccurs="5">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="component2"
minOccurs="0" maxOccurs="1">
<xsd:simpleType>
<xsd:restriction base="xsd:string">
<xsd:length value="8"/>
</xsd:restriction>
</xsd:simpleType>
</xsd:element>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
```

最上位のデータ構造は、前の例のデータ構造とまったく同じです。

```
05 component1-num PIC S9(9) COMP-5
05 component1-cont PIC X(16)
```

ただし、2 番目のデータ構造には、以下のエレメントが格納されます。

```
01 DFHWS-component1
02 component2-num PIC S9(9) COMP-5
02 component2-cont PIC X(16)
```

3 番目の構造には以下のエレメントが格納されます。

```
01 DFHWS-component2
02 component2 PIC X(8)
```

最外部エレメント <component1> の出現回数は、component1-num に格納されます。

component1-cont で指定されたコンテナーには、2 番目のデータ構造 (DFHWS-component1) のその数のインスタンスを持つ配列が含まれています。

component2-cont の各インスタンスは、異なるコンテナーを指定します。それぞれ、第 3 レベルの構造 (DFHWS-component2) によってマップされたデータ構造が含まれています。

この構造を説明するために、例と一致する次のような XML のフラグメントについて考えてみます。

```
<component1><component2>
string1
</component2></component1>
<component1><component2>
string2
</component2></component1>
<component1></component1>
```

<component1> は 3 回発生します。 最初の 2 つには、それぞれ <component2> のインスタンスが含まれていますが、3 番目のインスタンスには含まれていません。

最上位のデータ構造では、component1-num に値 3 が含まれています。 component1-cont で指定されたコンテナーには、DFHWS-component1 の次の 3 つのインスタンスがあります。

1. 第 1 のインスタンスでは、component2-num の値は 1 で、component2-cont で指定されたコンテナーは *string1* を保持します。
2. 第 2 のインスタンスでは、component2-num の値は 1 で、component2-cont で指定されたコンテナーは *string2* を保持します。
3. 第 3 のインスタンスでは、component2-num の値は 0 で、component2-cont の内容は未定義です。

このインスタンスでは、完全なデータ構造は、次の合計 4 つのコンテナーによって表現されます。

- コンテナー DFHWS-DATA 内のルート・データ構造
- component1-cont で指定されたコンテナー
- component2-cont の最初の 2 つのインスタンスで指定された 2 つのコンテナー

オプションの構造および xsd:choice

DFHWS2LS および DFHSC2LS はマッピング・レベル 2.1 以上で、<xsd:sequence>、<xsd:choice>、および <xsd:all> エレメントでの maxOccurs および minOccurs の使用をサポートします。この場合、minOccurs 属性および maxOccurs 属性は minOccurs="0" および maxOccurs="1" に設定されています。

アシスタントは、エレメント内の子エレメントがそれぞれオプションであるかのようにしてエレメントを処理するマッピングを生成します。 これらのエレメントを使用してアプリケーションを実装する場合、アプリケーションが無効なオプションの組み合わせを生成しないように確認してください。 それぞれのエレメントの生成された言語構造には count フィールドがありますが、これらのすべてが「0」に設定されるか、すべてが「1」に設定される必要があります。これ以外の値の組み合わせは、<xsd:choice> エレメントの場合を除き、無効です。

<xsd:choice> エレメントは、そのエレメントのオプションが 1 つのみ使用できることを示します。 これはすべてのマッピング・レベルでサポートされます。 アシスタントは <xsd:choice> のそれぞれのオプションを、minOccurs="0" および maxOccurs="1" である <xsd:sequence> エレメントのように扱います。<xsd:choice> エレメントを使用してアプリケーションを実装する場合は、アプリケーションが無効なオプションの組み合わせを生成しないように確認してください。それぞれのエレメントの生成された言語構造には count フィールドがありますが、そのうちの 1 つが「1」に、他のすべてが「0」に設定される必要があります。 これ以外の組み合わせはすべて無効です。例外は、<xsd:choice> のエレメント自体がオプションの場合であり、すべてのフィールドが 0 に設定されていても有効です。

XML 属性のサポート

XML スキーマは、XML で許可される属性または必須の属性を指定することができます。 CICS アシスタント・ユーティリティーである DFHWS2LS および DFHSC2LS はデフォルトで XML 属性を無視します。 XML スキーマに定義されている XML 属性を処理するには、**MAPPING-LEVEL** パラメーターの値が 1.1 以上である必要があります。

オプションの属性

属性にはオプションの属性と必須属性があり、SOAP メッセージ内またはアプリケーションの XML 内の任意のエレメントに関連付けることができます。 スキーマで定義されたすべてのオプションの属性ごとに、次に示す 2 つのフィールドが適切な言語構造で生成されます。

1. 存在フラグ。このフィールドはブール・データ・タイプとして扱われ、通常は長さが 1 バイトです。
2. 値。このフィールドは、同等のタイプの XML エレメントと同じ方法でマップされます。 例えば、タイプ NMTOKEN の属性は、タイプ NMTOKEN の XML エレメントと同じようにマップされます。

属性の存在フィールドと値フィールドは、関連付けられたエレメントのフィールドの前に、生成された言語構造で表示されます。 インスタンス文書に現れる予期しない属性は無視されます。

例えば、次のスキーマ属性定義について考えてみます。

```
<xsd:attribute name="age" type="xsd:short"
use="optional" />
```

このオプションの属性は次に示す COBOL 構造にマップされます。

```
05 attr-age-exist PIC X DISPLAY
05 attr-age-value PIC S9999 COMP-5 SYNC
```

オプションの属性のランタイム処理

オプションの属性では次に示すランタイム処理が実行されます。

- 属性が存在する場合は、存在フラグが設定され、値はマップされます。
- 属性が存在しない場合は、存在フラグは設定されません。
- 属性がデフォルト値を持ち、存在する場合は、値がマップされます。
- 属性がデフォルト値を持ち、存在しない場合は、デフォルト値がマップされます。

デフォルト値を持つオプションの属性は、必須属性として扱われます。

CICS がデータを XML に変換すると、次のランタイム処理が行われます。

- 存在フラグが設定される場合は、属性は変換されて XML に含められます。
- 存在フラグが設定されない場合は、属性は XML に含められません。

必須属性とランタイム処理

すべての必須属性で、値フィールドのみが適切な言語構造で生成されます。

属性が XML に存在する場合は、値がマップされます。 属性が存在しない場合、次に示す処理が行われます。

- アプリケーションが Web サービス・プロバイダーの場合、CICS はクライアントの SOAP メッセージ内に、エラーを示す SOAP 障害メッセージを生成します。

- アプリケーションが Web サービス要求側の場合、CICS はメッセージを発行して、13 の RESP2 コード付きの変換エラー応答をアプリケーションに返します。
- アプリケーションが **TRANSFORM XMLTODATA** コマンドを使用している場合、CICS はメッセージを発行し、無効な要求応答 (RESP2 コード 3) をアプリケーションに戻します。

CICS が COMMAREA またはコンテナーの内容を基にして SOAP メッセージを生成すると、属性は変換されて、メッセージに含められます。 アプリケーションが **TRANSFORM DATATOXML** コマンドを使用する場合、CICS は属性の変換も行って XML に含めます。

nillable 属性

nillable 属性とは、XML スキーマの xsd:element で指定される場合のある特殊な属性です。これは、XML のエレメントで xsi:nil 属性が有効であることを指定します。エレメントで xsi:nil 属性が指定されている場合、エレメントは存在するが値がないため、このエレメントには内容が関連付けられていないことを示します。

XML スキーマが nillable 属性を true として定義した場合は、ブール値を使用する必須属性としてマップされます。

CICS が xsi:nil 属性を含む SOAP メッセージを受信した場合や、この属性を含む XML をアプリケーション用に変換する必要がある場合、属性の値は true または false です。値が true の場合は、xsi:nil 属性のスコープ内のエレメントまたはネストされたエレメントは、アプリケーションによって無視される必要があります。

CICS が、xsi:nil 属性の値が true である COMMAREA またはコンテナーの内容を基にして SOAP メッセージまたは XML を生成すると、次に示す処理が行われます。

- xsi:nil 属性が XML または SOAP メッセージに生成されます。
- 関連するエレメントの値は無視されます。
- エレメント内でネストされたエレメントはすべて無視されます。

SOAP メッセージの例

WSDL 文書の一部になる場合がある次の例の XML スキーマについて考えてみます。

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<xsd:element name="root" nillable="true">
<xsd:complexType>
<xsd:sequence>
<xsd:element nillable="true" name="num" type="xsd:int" maxOccurs="3"
minOccurs="3"/>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:schema>
```

このスキーマに適合する部分的な SOAP メッセージの例を、次に示します。

```
<root xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<num xsi:nil="true"/>
<num>15</num>
<num xsi:nil="true"/>
</root>
```

COBOL では、この SOAP メッセージは次のエレメントにマップされます。

```
05 root
10 attr-nil-root-value PIC X DISPLAY
10 num OCCURS 3
15 num1 PIC S9(9) COMP-5 SYNC
15 attr-nil-num-value PIC X DISPLAY
10 filler PIC X(3)
```

<xsd:any> および xsd:anyType のサポート

DFHWS2LS および DFHSC2LS は XML スキーマにおける <xsd:any> および xsd:anyType の使用をサポートしています。 <xsd:any> XML スキーマ・エレメントを使用して未定義の内容を持つ XML 文書のセクションを記述できます。 xsd:anyType とは、すべての単純および複合データ・タイプの派生元となる基本のデータ・タイプです。データ内容に制限や制約はありません。

CICS アシスタントで <xsd:any> および xsd:anyType を使用する前に、次に示すパラメーターを設定してください。

- **MAPPING-LEVEL** パラメーターを 2.1 以上に設定します。
- Web サービス・プロバイダー・アプリケーションの場合、**PGMINT** パラメーターを CHANNEL に設定します。

<xsd:any> の例

この例では <xsd:any> エレメントを使用して、"Customer" タグの "Surname" タグの後に、オプションの非構造化 XML コンテンツを記述しています。

```
<xsd:element name="Customer">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="Title" type="xsd:string"/>
<xsd:element name="FirstName" type="xsd:string"/>
<xsd:element name="Surname" type="xsd:string"/>
<xsd:any minOccurs="0"/>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
```

この XML スキーマに適合する SOAP メッセージの例を以下に示します。

```
<xml version='1.0' encoding='UTF-8'?>
    <SOAP-ENV:Envelope
    xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Body>
    <Customer xmlns="http://www.example.org/anyExample">
    <Title xmlns="">Mr</Title>
    <FirstName xmlns="">John</FirstName>
    <Surname xmlns="">Smith</Surname>
    <ExtraInformation xmlns="http://www.example.org/ExtraInformation">
    <!-- This 'ExtraInformation' tag is associated with the optional xsd:any
    from the XML schema.
    It can contain any well formed XML. -->
    <ExampleField1>one</ExampleField1>
    <ExampleField2>two</ExampleField2>
```

```
    </ExtraInformation>
    </Customer>
    </SOAP-ENV:Body>
    </SOAP-ENV:Envelope>
```

この SOAP メッセージが CICS に送信されると、CICS は Customer-xml-cont コンテナーに次に示す XML データを追加します。

```
<ExtraInformation
    xmlns="http://www.example.org/ExtraInformation">
    <!-- This 'ExtraInformation' tag is associated with the optional xsd:any
    from the XML schema.
    It can contain any well formed XML. -->
    <ExampleField1>one</ExampleField1>
    <ExampleField2>two</ExampleField2>
    </ExtraInformation>
```

CICS はまた Customer-xmlns-cont コンテナーに、次に示すスコープ内の XML 名前空間宣言を追加します。これらの宣言はスペースで区切られています。

```
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns="http://www.example.org/anyExample"
```

xsd:anyType の例

xsd:anyType とは、すべての単純および複合データ・タイプの派生元となる基本のデータ・タイプです。 これはデータ内容を制限しません。 データ・タイプを指定しないと、デフォルトとして xsd:anyType に設定されます。例えば、これらの 2 つの XML フラグメントは等価です。

```
<xsd:element name="Name" type="xsd:anyType"/>
```

```
<xsd:element name="Name"/>
```

生成される言語構造

< xsd:any > または xsd:anyType 用に生成される言語構造の形式は、COBOL では次のようになります。他の言語ではこれに相当する形式になります。

elementName-xml-cont PIC X(16)

未加工の XML を格納するコンテナーの名前。 CICS は、着信 SOAP メッセージを処理する際、<xsd:any> または xsd:anyType が定義する SOAP メッセージのサブセットをこのコンテナーに置きます。 アプリケーションは XML データをネイティブでのみ処理できます。アプリケーションは XML を生成し、このコンテナーにデータを取り込み、コンテナー名を提供する必要があります。

このコンテナーへの取り込みはテキスト・モードで行われる必要があります。 CICS がこのコンテナーにデータを取り込む場合、Web サービスが使用するように定義された EBCDIC のバリアントと同じものを使用して取り込みが行われます。ターゲットの EBCDIC コード・ページに存在しない文字は置換文字によって置き換えられます。アプリケーションによってコンテナーが UTF-8 で読み取られる場合も同様です。

elementName-xmlns-cont PIC X(16)

スコープ内の任意の名前空間の接頭部宣言を格納しているコンテナーの名前。 SOAP エンベロープ・タグのサブセットだけではなく、スコープ内にあり関連性のある名前空間の宣言がすべて含まれている点を除けば、このコンテナーの内容は DFHWS-XMLNS コンテナーの内容と似ています。

このコンテナーへの取り込みはテキスト・モードで行われる必要があります。 CICS がこのコンテナーにデータを取り込む場合、Web サービスが使用するように定義された EBCDIC のバリアントと同じものを使用して取り込みが行われます。ターゲットの EBCDIC コード・ページに存在しない文字は置換文字によって置き換えられます。アプリケーションによってコンテナーが UTF-8 で読み取られる場合も同様です。

このコンテナーは CICS に送信された SOAP メッセージを処理するときにのみ使用されます。出力 SOAP メッセージの生成時にアプリケーションがコンテナーに名前空間宣言を提供しようとする場合、コンテナーとその内容は CICS によって無視されます。 CICS では、アプリケーションが提供する XML に名前空間宣言が含まれていることが必要とされます。

<xsd:any> エレメントを含んでいる XML エレメントの名前は、<xsd:any> エレメント用に生成される変数名に組み込まれます。 <xsd:any> の例では、<xsd:any> エレメントは <xsd:element name="Customer"> エレメント内にネストされており、<xsd:any> エレメント用に生成される変数名は `Customer-xml-cont PIC X(16)` および `Customer-xmlns-cont PIC X(16)` です。

`xsd:anyType` タイプの場合、直接の XML エレメント名が使用されます。上記の `xsd:anyType` の例では、変数名は `Name-xml-cont PIC X(16)` および `Name-xmlns-cont PIC X(16)` です。

<xsd:choice> のサポート

<xsd:choice> エレメントは、そのエレメントでオプションが 1 つだけ使用できることを示します。 CICS アシスタントは、さまざまなマッピング・レベルの <xsd:choice> エレメントに、さまざまな度合いのサポートを提供しています。

マッピング・レベル 2.2 以上の <xsd:choice> のサポート

マッピング・レベル 2.2 以上では、DFHWS2LS および DFHSC2LS の <xsd:choice> エレメントのサポートが改善されています。 アシスタントは <xsd:choice> エレメントに関連した値を格納する新しいコンテナーを生成します。アシスタントは、以下に示すように新規コンテナー名および追加のフィールド名を含む言語構造を生成します。

***fieldname* -enum**
: <xsd:choice> エレメントが使用するオプションを区別して示すフィールド。

***fieldname* -cont**
: 使用されるオプションを格納するコンテナーの名前。 オプションの値をマップするために、さらに言語構造が生成されます。

次に示す XML スキーマのフラグメントには <xsd:choice> エレメントが含まれています。

```
<xsd:element name="choiceExample">
<xsd:complexType>
<xsd:choice>
<xsd:element name="option1" type="xsd:string" />
<xsd:element name="option2" type="xsd:int" />
```

```
<xsd:element name="option3" type="xsd:short" maxOccurs="2" minOccurs="2" />
</xsd:choice>
</xsd:complexType>
</xsd:element>
```

この XML スキーマ・フラグメントがマッピング・レベル 2.2 以上で処理される場合、アシスタントは次に示す COBOL 言語構造を生成します。

```
 03 choiceExample.
06 choiceExample-enum PIC X DISPLAY.
88 empty VALUE X'00'.
88 option1 VALUE X'01'.
88 option2 VALUE X'02'.
88 option3 VALUE X'03'.
06 choiceExample-cont PIC X(16).


01 Example-option1.
03 option1-length PIC S9999 COMP-5 SYNC.
03 option1 PIC X(255).

01 Example-option2.
03 option2 PIC S9(9) COMP-5 SYNC.

01 Example-option3.
03 option3 OCCURS 2 PIC S9999 COMP-5 SYNC.
```

マッピング・レベル 2.2 以上における <xsd:choice> の制限

DFHSC2LS および DFHWS2LS はネストされた <xsd:choice> エレメントをサポートしません。例えば、次に示す XML はサポートされません。

```
<xsd:choice>
<xsd:element name ="name1" type="string"/>
<xsd:choice>
<xsd:element name ="name2a" type="string"/>
<xsd:element name ="name2b" type="string"/>
</xsd:choice>
</xsd:choice>
```

DFHSC2LS および DFHWS2LS は循環する <xsd:choice> エレメントをサポートしません。例えば、次の XML はサポートされません。

```
<xsd:choice maxOccurs="2">
<xsd:element name ="name1" type="string"/>
</xsd:choice>
```

DFHSC2LS および DFHWS2LS では、1 つの <xsd:choice> エレメントで最大 255 までのオプションをサポートします。

マッピング・レベル 2.1 以下の <xsd:choice> のサポート

2.1 以下のマッピング・レベルでは、DFHWS2LS が <xsd:choice> エレメントに関して提供するサポートが限定されます。 DFHWS2LS は <xsd:choice> エレメントのそれぞれのオプションを、最大で 1 回生じる可能性のある <xsd:sequence> エレメントのように扱います。

<xsd:choice> エレメントのオプションのうち 1 つしか使用できないため、<xsd:choice> エレメントを使用してアプリケーションを実装する場合は有効なオプションの組み合わせのみが生成されるかどうかに注意してください。 それぞれのエレメントの生成された言語構造には count フィールドがありますが、そのうちの 1

つが 1 に、他のすべてが 0 に設定される必要があります。これ以外の組み合わせはすべて無効です。例外は、<xsd:choice> 自体がオプションの場合であり、すべてのフィールドが 0 に設定されても有効です。

<xsd:sequence> のサポート

CICS アシスタントを使用した <xsd:sequence> 要素の使用がサポートされていますが、いくつかの制約があります。

minOccurs 属性および maxOccurs 属性は、<xsd:sequence> 要素ではサポートされていません。この規則の例外は、minOccurs="0" および maxOccurs="1" または minOccurs="1" および maxOccurs="1" の場合です。

CICS アシスタントでは、同じ名前の 2 つの要素を同じ <xsd:sequence> 要素で使用することはできません。 例えば、シーケンス {a, b, c, a} は、CICS アシスタントによって拒否されます。 この制限を回避するには、シーケンスを {a maxOccurs="2", b, c} で置き換えます。

CICS アシスタントでは、同じ <xsd:sequence> 内に、<xsd:any> 要素が 1 つのみ、または <xsd:any> として処理される要素が 1 つのみ許可されます。<xsd:any> 要素として処理される要素には、定義された置換グループを持たない抽象要素が含まれます。

XML を構文解析する場合、CICS は <xsd:sequence> 要素を <xsd:all> 要素のように処理します。このことは、CICS ではシーケンス内の項目の順序が正しいかどうかは確認されないことを意味します。 例えば、スキーマが {a, b, c} のシーケンスを定義する場合、CICS は a, b, および c を任意の順序で含む XML を許容します。 ただし、CICS が XML を生成する際には、<xsd:sequence> の項目の順序は常に保存されます。

CICS では、欠落した XML データは自動的に検出されません。 例えば、<xsd:sequence> の要素が必須であると定義されている (minOccurs="1" maxOccurs="1") ものの、XML 文書に出現しない場合、CICS はこの問題を実行時の妥当性検査が使用可能になっている場合にのみ報告します。

置換グループのサポート

置換グループを使用して、交換可能な XML エレメントのグループを定義できます。 CICS アシスタントはマッピング・レベル 2.2 以上で置換グループをサポートしています。

マッピング・レベル 2.2 以上では、DFHSC2LS および DFHWS2LS は <xsd:choice> エレメントに使用するマッピングに似たマッピングを使用して置換グループをサポートします。アシスタントは列挙フィールドと新規のコンテナー名を、その言語構造で生成します。

次に示す XML スキーマ・フラグメントには 2 つの subGroupParent エレメントの配列が含まれており、それぞれ replacementOption1 または replacementOption2 と置換可能です。

```
<xsd:element name="subGroupExample"
   >
   <xsd:complexType>
   <xsd:sequence>
   <xsd:element ref="subGroupParent" maxOccurs="2" minOccurs="2" />
```

```
</xsd:sequence>
</xsd:complexType>
</xsd:element>

<xsd:element name="subGroupParent" type="xsd:anySimpleType" />
<xsd:element name="replacementOption1" type="xsd:int"
substitutionGroup="subGroupParent" />
<xsd:element name="replacementOption2" type="xsd:short"
substitutionGroup="subGroupParent" />
```

この XML フラグメントをアシスタントで処理すると、次に示す COBOL 言語構造が生成されます。

```
03 subGroupExample.
06 subGroupParent OCCURS2.
09 subGroupExample-enum PIC X DISPLAY.
88 empty VALUE X '00'.
88 replacementOption1 VALUE X '01'.
88 replacementOption2 VALUE X '02'.
88 subGroupParent VALUE X '03'.
09 subGroupExample-cont PIC X (16).



01 Example-replacementOption1.
03 replacementOption1 PIC S9(9) COMP-5 SYNC.

01 Example-replacementOption2.
03 replacementOption2 PIC S9999 COMP-5 SYNC.

01 Example-subGroupParent.
03 subGroupParent-length PIC S9999 COMP-5 SYNC.
03 subGroupParent PIC X(255).
```

置換グループについて詳しくは、XML Schema Part 1: Structures Second Edition を参照してください。

抽象エレメントおよび抽象データ・タイプのサポート

CICS アシスタントはマッピング・レベル 2.2 以上で抽象エレメントおよび抽象データ・タイプをサポートします。 CICS アシスタントは、置換グループと同様の方法で抽象エレメントおよび抽象データ・タイプをマップします。

マッピング・レベル 2.2 以上における抽象エレメントのサポート

マッピング・レベル 2.2 以上では、DFHSC2LS および DFHWS2LS は、抽象エレメントが置換グループの有効なメンバーではないことを除くと、抽象エレメントを置換グループとほぼ同じ方法で扱います。 置換可能なエレメントがない場合、抽象エレメントは <xsd:any> エレメントとして扱われ、マッピング・レベル 2.1 における <xsd:any> エレメントと同じマッピングを使用します。

次に示す XML スキーマ・フラグメントは、抽象エレメントの代わりに使用できる 2 つのオプションを指定します。 抽象エレメント自体は有効なオプションではありません。

```
<xsd:element name="abstractElementExample" >
<xsd:complexType>
<xsd:sequence>
<xsd:element ref="abstractElementParent" maxOccurs="2" minOccurs="2" />
</xsd:sequence>
</xsd:complexType>
</xsd:element>
```

```
<xsd:element name="abstractElementParent" type="xsd:anySimpleType"
abstract="true" />
<xsd:element name="replacementOption1" type="xsd:int"
substitutionGroup="abstractElementParent" />
<xsd:element name="replacementOption2" type="xsd:short"
substitutionGroup="abstractElementParent" />
```

この XML フラグメントをアシスタントで処理すると、次に示す COBOL 言語構造が生成されます。

```
 03 abstractElementExample.
06 abstractElementParent OCCURS 2.
09 abstractElementExample-enum PIC X DISPLAY.
88 empty VALUE X '00'.
88 replacementOption1 VALUE X '01'.
88 replacementOption2 VALUE X '02'.
09 abstractElementExample-cont PIC X (16).


01 Example-replacementOption1.
03 replacementOption1 PIC S9(9) COMP-5 SYNC.

01 Example-replacementOption2.
03 replacementOption2 PIC S9999 COMP-5 SYNC.
```

抽象エレメントについて詳しくは、XML Schema Part 0: Primer Second Editionを参照してください。

マッピング・レベル 2.2 以上における抽象データ・タイプのサポート

マッピング・レベル 2.2 以上では、DFHSC2LS および DFHWS2LS が抽象データ・タイプを置換グループとして扱います。 アシスタントは列挙フィールドと新規のコンテナー名を、その言語構造で生成します。

次に示す XML スキーマ・フラグメントは、抽象タイプの代わりに使用できる 2 つのオプションを指定します。

```
<xsd:element name="AbstractDataTypeExample"
type="abstractDataType" />

<xsd:complexType name="abstractDataType" abstract="true">
<xsd:simpleContent>
<xsd:extension base="xsd:string" />
</xsd:simpleContent>
</xsd:complexType>
<xsd:complexType name="option1">
<xsd:simpleContent>
<xsd:restriction base="abstractDataType">
<xsd:length value="5" />
</xsd:restriction>
</xsd:simpleContent>
</xsd:complexType>
<xsd:complexType name="option2">
<xsd:simpleContent>
<xsd:restriction base="abstractDataType">
<xsd:length value="10" />
</xsd:restriction>
</xsd:simpleContent>
</xsd:complexType>
```

この XML フラグメントをアシスタントで処理すると、次に示す COBOL 言語構造が生成されます。

```
 03 AbstractDataTypeExamp-enum PIC X DISPLAY.
88 empty VALUE X'00'.
88 option1 VALUE X'01'.
88 option2 VALUE X'02'.
03 AbstractDataTypeExamp-cont PIC X(16).
```

言語構造は別々のコピーブックに生成されます。 option1 用に生成される言語構造は、以下のコピーブックに生成されます。

```
 03 option1 PIC X(5).
```

option2 用に生成される言語構造は、以下の別のコピーブックに生成されます。

```
 03 option2 PIC X(10).
```

抽象データ・タイプについて詳しくは、XML Schema Part 0: Primer Second Editionを参照してください。

COBOL における可変の繰り返しコンテンツの処理方法

COBOL では、データの各インスタンスを扱うポインター演算を使用して、可変繰り返しコンテンツを処理することはできません。他のプログラミング言語にはこの制限はありません。 この例では、Web サービス・アプリケーションのために COBOL で可変の繰り返しコンテンツを処理する方法を示します。

この方法は、**TRANSFORM** API コマンドを使用する XML のアプリケーション・データへの変換にも適用できます。 次の WSDL 文書の例は、繰り返し回数が可変である 8 文字のストリングからなるアプリケーション・データを含む Web サービスを表します。

```
<?xml version="1.0"?>
     <definitions name="ExampleWSDL"
     targetNamespace="http://www.example.org/variablyRepeatingData/"
     xmlns="http://schemas.xmlsoap.org/wsdl/"
     xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
     xmlns:tns="http://www.example.org/variablyRepeatingData/"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema">

     <types>
     <xsd:schema targetNamespace="http://www.example.org/variablyRepeatingData/">
     <xsd:element name="applicationData">
     <xsd:complexType>
     <xsd:sequence>
     <xsd:element name="component" minOccurs="1" maxOccurs="unbounded">
     <xsd:simpleType>
     <xsd:restriction base="xsd:string">
     <xsd:length value="8"/>
     </xsd:restriction>
     </xsd:simpleType>
     </xsd:element>
     </xsd:sequence>
     </xsd:complexType>
     </xsd:element>
     </xsd:schema>
     </types>

     <message name="exampleMessage">
     <part element="tns:applicationData" name="messagePart"/>
     </message>

     <portType name="examplePortType">
     <operation name="exampleOperation">
     <input message="tns:exampleMessage"/>
     <output message="tns:exampleMessage"/>
     </operation>
```

```
    </portType>

    <binding name="exampleBinding" type="tns:examplePortType">
    <soap:binding style="document"
    transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="exampleOperation">
    <soap:operation soapAction=""/>
    <input><soap:body parts="messagePart" encodingStyle="" use="literal"/></input>
    <output><soap:body parts="messagePart" encodingStyle=""
    use="literal"/></output>
    </operation>
    </binding>
    </definitions>
```

この WSDL 文書を DFHWS2LS を使用して処理すると、次に示す COBOL 言語構造が生成されます。

```
 03 applicationData.

06 component-num PIC S9(9) COMP-5 SYNC.
06 component-cont PIC X(16).


01 DFHWS-component.
03 component PIC X(8).
```

8 文字の component フィールドが、DFHWS-component という名前の別の構造で定義されていることに注意してください。メインのデータ構造は applicationData と呼ばれており、component-num および component-cont という 2 つのフィールドを含んでいます。 component-num フィールドは、component データのインスタンス数を示し、component-cont フィールドは component フィールドの連結リストを格納するコンテナーの名前を示します。

次に示す COBOL コードは、可変の繰り返しデータのリストを処理する方法の 1 つを例示しています。ここでは linkage section 配列を使用してデータの後続インスタンスを処理しています。インスタンスはそれぞれ DISPLAY ステートメントを使用して表示されます。

```
IDENTIFICATION DIVISION.
PROGRAM-ID. EXVARY.

ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.

* working storage variables
01 APP-DATA-PTR USAGE IS POINTER.
01 APP-DATA-LENGTH PIC S9(8) COMP.
01 COMPONENT-PTR USAGE IS POINTER.
01 COMPONENT-DATA-LENGTH PIC S9(8) COMP.
01 COMPONENT-COUNT PIC S9(8) COMP-4 VALUE 0.
01 COMPONENT-LENGTH PIC S9(8) COMP.

LINKAGE SECTION.

* a large linkage section array
01 BIG-ARRAY PIC X(659999).

* application data structures produced by DFHWS2LS
* this is normally referenced with a COPY statement
01 DFHWS2LS-data.
03 applicationData.
06 component-num PIC S9(9) COMP-5 SYNC.
06 component-cont PIC X(16).
```

```
01 DFHWS-component.
03 component PIC X(8).



PROCEDURE DIVISION USING DFHEIBLK.
A-CONTROL SECTION.
A010-CONTROL.

* Get the DFHWS-DATA container
EXEC CICS GET CONTAINER('DFHWS-DATA')
SET(APP-DATA-PTR)
FLENGTH(APP-DATA-LENGTH)
END-EXEC
SET ADDRESS OF DFHWS2LS-data TO APP-DATA-PTR

* Get the recurring component data
EXEC CICS GET CONTAINER(component-cont)
SET(COMPONENT-PTR)
FLENGTH(COMPONENT-DATA-LENGTH)
END-EXEC

* Point the component structure at the first instance of the data
SET ADDRESS OF DFHWS-component TO COMPONENT-PTR

* Store the length of a single component
MOVE LENGTH OF DFHWS-component TO COMPONENT-LENGTH

* process each instance of component data in turn
PERFORM WITH TEST AFTER
UNTIL COMPONENT-COUNT = component-num

* display the current instance of the data
DISPLAY 'component value is: ' component

* address the next instance of the component data
SET ADDRESS OF BIG-ARRAY TO ADDRESS OF DFHWS-component
SET ADDRESS OF DFHWS-component
TO ADDRESS OF BIG-ARRAY (COMPONENT-LENGTH + 1:1)
ADD 1 TO COMPONENT-COUNT

* end the loop
END-PERFORM.

* Point the component structure back at the first instance of
* of the data, for any further processing we may want to perform
SET ADDRESS OF DFHWS-component TO COMPONENT-PTR

* return to CICS.

EXEC CICS
RETURN
END-EXEC

GOBACK.
```

上記のコードは可変の繰り返しコンテンツの一般的な処理方法を提供しています。配列 BIG-ARRAY はそれぞれのコンポーネントの先頭に順次移動し、データの先頭に固定されません。 コンポーネントのデータ構造は、次のコンポーネントの最初のバイトを指すように移動します。 COMPONENT-PTR を使用して、必要に応じてコンポーネント・データの開始位置を回復できます。

WSDL 文書に準拠する SOAP メッセージの例を次に示します。

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Body>
<applicationData xmlns="http://www.example.org/variablyRepeatingData/">
<component xmlns="">VALUE1</component>
<component xmlns="">VALUE2</component>
<component xmlns="">VALUE3</component>
</applicationData>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

COBOL プログラムが SOAP メッセージを処理する際に生成される出力は次のとおりです。

```
CPIH 20080115103151 component value is: VALUE1
CPIH 20080115103151 component value is: VALUE2
CPIH 20080115103151 component value is: VALUE3
```

可変長の値と空白のサポート

CICS 支援機能の設定値を使用したり、XML スキーマに直接ファセットを追加したりすることにより、可変長の値と空白を処理する方法をカスタマイズできます。

通常、CICS XML 支援機能と CICS Web サービス支援機能は、データ・ストリングを固定長の文字配列にマップします。こうした配列では、スペースや NULL で埋め込む必要があります。 可変長の値を固定長のデータ配列にマッピングすると、効率が悪く、ストレージを浪費する恐れがあります。使用するデータ長が可変の場合、こうしたマッピングを処理する方法をカスタマイズすることが推奨されています。

ある言語構造から XML スキーマまたは WSDL 文書に変換する場合には、XML スキーマで `whiteSpace` ファセットと `maxLength` ファセットを指定し、支援機能で **`CHAR-VARYING-LIMIT`** パラメーターを設定することが推奨されています。

XML スキーマまたは WSDL 文書からある言語構造に変換する場合には、支援機能で **`CHAR-VARYING`** パラメーターを適切な値に設定することが推奨されています。

注: XML 文書では、NULL 文字 ('x00') は無効です。 CICS が構文解析するアプリケーション・データの NULL 文字はストリングの末尾を示すためのもので、値は切り捨てられます。 CICS は、アプリケーション・データを生成する際に、**`CHAR-VARYING`** パラメーターの値に従って生成します。 例えば、**`CHAR-VARYING=NULL`** オプションが指定されている場合、CICS によって生成される可変長のストリングはヌル文字で終わります。

XML から言語構造への可変長の値のマッピング

XML スキーマのファセットを使用して、または CICS 支援機能で特定のパラメーターを指定して、XML スキーマまたは WSDL 文書と言語構造間のマッピングを処理する方法をカスタマイズします。

ファセットを使用して XML データ・タイプを制限することができます。 XML 内の可変長データの処理方法をカスタマイズするには、長さファセット (`length`、`maxLength`、および `minLength`) および `whiteSpace` ファセットを使用します。

length
データが固定長となるように指定します。

maxLength
データ・タイプの最大長を指定します。 ストリング・ベースのデータ・タイプでこの値を設定しないと、最大長は無制限になります。

minLength
データ・タイプの最小長を指定します。 ストリング・ベースのデータ・タイプでこの値を設定しないと、最小長は 0 になります。

whiteSpace
データ値の周囲にある空白の処理方法を指定します。空白には、スペース、タブ、および改行が含まれます。 whiteSpace ファセットは、preserve、replace、または collapse に設定できます。

- 値 preserve は、データ値内の空白をすべて保持します。
- 値 replace は、タブまたは改行が適切な数のスペースに置換されることを意味します。
- 値 collapse は、先行空白、後続空白、および埋め込み空白は除去され、タブ、改行、および連続スペースはすべて単一のスペース文字に置き換えられることを意味します。

whiteSpace ファセットが設定されないと、空白は保持されます。

XML スキーマ・ファセットについて詳しくは、XML Schema Part 2: Datatypes Second Editionを参照してください。

CICS 支援機能でパラメーター DFHSC2LS および DFHWS2LS を使用すると、XML スキーマから言語構造への可変長データのマッピング方法を変更できます。これらのパラメーターは、マッピング・レベル 1.2 以降で使用可能です。

DEFAULT-CHAR-MAXLENGTH
XML スキーマ文書または WSDL 文書で長さが指定されていない場合に、マッピングする文字の文字データのデフォルトの配列長を指定します。 このパラメーターの値は、1 から 2 147 483 647 の範囲の正整数です。

ただし、XML スキーマまたは WSDL 文書で maxLength ファセットを使用して、DFHSC2LS または DFHWS2LS が使用する最大文字長を直接指定することをお勧めします。 XML スキーマまたは WSDL 文書で最大長を直接指定すると、すべてのストリング・ベースのデータ・タイプに 1 つのグローバル・デフォルトが適用されることに関連した問題を回避できます。

CHAR-VARYING-LIMIT
言語構造にマップされる可変長文字データの最大サイズを指定します。 文字データが、このパラメーターで指定されている値よりも大きい場合は、コンテナーにマップされ、生成された言語構造でそのコンテナー名が使用されます。値は 0 からデフォルトの 32 767 バイトの範囲で指定できます。

CHAR-VARYING
可変長文字データがどのようにマップされるのかを指定します。 このパラメーターを指定しない場合は、指定される言語に応じてデフォルトのマッピングが異なります。 以下のオプションを選択できます。

- **CHAR-VARYING=NO** は、可変長文字データが固定長ストリングとしてマップされるように指定します。
- **CHAR-VARYING=NULL** は、可変長文字データがヌル終了ストリングにマップされるように指定します。
- **CHAR-VARYING=YES** は、可変長文字データが PL/I で CHAR VARYING データ・タイプにマップされるように指定します。 COBOL、C、および C++ 言語の場合、可変長文字データは、2 つの関連エレメント (データ長およびデータ) で構成される同等の表現にマップされます。

通常、**CHAR-VARYING=YES** に設定すると、最高のパフォーマンスが得られます。

言語構造から XML への可変長の値のマッピング

ご使用の言語構造と XML スキーマまたは WSDL 文書間におけるマッピングの処理方法をカスタマイズできます。 DFHLS2SC または DFHLS2WS の **CHAR-VARYING** パラメーターを COLLAPSE または NULL に設定して、文字配列が生成される方法を変更します。

CHAR-VARYING=NULL オプションを設定すると、CICS に XML を生成する際に各文字配列の末尾に NULL 文字を追加するように指示することになります。

CHAR-VARYING=COLLAPSE オプションを設定すると、CICS に XML を生成する際に文字配列の末尾から後続スペースを自動的に除去するように指示することになります。 このオプションが有効なのはマッピング・レベルが 2.1 以上の場合だけで、C および C++ 以外のすべての言語ではマッピング・レベル 2.1 以降のデフォルト値が **CHAR-VARYING=COLLAPSE** になります。 XML が解析されると、先行空白、後続空白、埋め込み空白はすべて削除されます。

アプリケーション・データでの UTF-16 サポート

CICS Web サービスでは、UTF-16 でエンコードされたアプリケーション・データから XML または JSON への変換と、XML または JSON から UTF-16 エンコード・アプリケーション・データへの変換がサポートされます。複数の言語でデータを保管および処理する必要がある場合には UTF-16 を使用してください。

CICS SOAP および JSON Web サービスでは、UTF-16 でエンコードされたアプリケーション・データから XML または JSON への変換と、XML または JSON から UTF-16 でエンコードされたアプリケーション・データへの変換がサポートされます。可変幅エンコード・スキームである Unicode を使用すると、システムは効率的にデータを扱うことができます。

UTF-16 は Unicode の可変幅エンコード方式であり、各文字が 2 バイトまたは 4 バイトで表されます。 CICS Web サービスはアプリケーション・データ用に CCSID 1200 をサポートします。これは、IBM 専用領域を含む UTF-16 BE (ビッグ・エンディアン) です。この動作は、サポートされるすべての言語で UTF-16 サポートと整合しています。

UTF-16 はマッピング・レベル 4.0 以上でサポートされます。アシスタントでマッピング設定を使用して、アプリケーション・データの変換方法をカスタマイズする

ことができます。 XML マッピング・レベルについて詳しくは、CICS アシスタントのマッピング・レベルを参照してください。JSON マッピング・レベルについて詳しくは、CICS JSON アシスタントのマッピング・レベルを参照してください。

注: UTF-16 は EBCDIC エンコード方式に比べてより多くの処理時間を必要とし、ストレージ効率が低くなります。さらに、複数の種類のエンコード方式を混合すると、実行時に追加の処理が発生します。

XML または JSON スキーマから言語構造への UTF-16 のマッピング

UTF-16 のサポートは、Web サービスの作成方法に応じて異なります。 XML または JSON スキーマから言語構造へのマッピング (トップダウン・マッピングともいう) には、次のような特性があります。 UTF-16 が有効になっている場合、すべてのテキスト・フィールドは UTF-16 フィールドにマップされ、COBOL での数値表示データ型は EBCDIC としてマップされます。 UTF-16 を使用するには、DFHJS2LS、DFHSC2LS、または DFHWS2LS の CCSID パラメーターを 1200 に設定してください。

例えば、WSDL の中に以下の XML スキーマ断片が存在する場合、

```
<xsd:element name="myString" nillable="false">
<xsd:simpleType>
<xsd:restriction base="xsd:string">
<xsd:maxLength value="20"/>
</xsd:restriction>
</xsd:simpleType>
</xsd:element>
```

DFHWS2LS アシスタントは COBOL 言語構造で以下のフィールドを生成します。

```
myString PIC N(
20
) USAGE NATIONAL
```

Web サービス・アシスタントの CHAR-MULTIPLIER パラメーターを使用すると、アシスタントによって生成されるフィールドの長さを指定できます。

CHAR-MULTIPLIER

UTF-16 を使用する場合、 **CHAR-MULTIPLIER** パラメーターに有効な値は、2 または 4 のみで、 2 はデフォルト値です。

CHAR-MULTIPLIER = 2。 ここで、スキーマは maxlength *x* のストリングを表し、 PIC N(*x*) を生成します。 **CHAR-MULTIPLIER**=2 を設定した場合、UTF-16 ストリングでの代理ペアは使用不可にはなりませんが、フィールドに入る文字数が影響を受けます。

CHAR-MULTIPLIER = 4 は、PIC N(*2x*) を生成します。 **CHAR-MULTIPLIER**=4 である場合、1 つのエンコード・ユニットで表現できる文字がストリングに含まれるならば、実行時の値が埋め込まれます。

言語構造から XML または JSON スキーマへの UTF-16 のマッピング

言語構造から XML または JSON スキーマへのマッピング (ボトムアップ・マッピングともいう) は、トップダウン・マッピングとは異なる方法で管理されます。言語構造で UTF-16 ストリングが宣言されている場合、CICS はデータが UTF-16 でエンコードされていると解釈し、そうでない場合は EBCDIC エンコード方式のデー

タを想定します。 DFHLS2JS、DFHLS2SC、または DFHLS2WS の CCSID パラメーターは、アプリケーション・データ内の EBCDIC テキストのエンコードを示します (UTF-16 を示すようこれを設定してはなりません)。

UTF-16 文字として解釈されるデータ型は、COBOL では `PIC N (n)`、PL/I では `WIDECHAR(n)`、C および C++ では `char16_t[n]` です。

Web サービス・アシスタントの CHAR-USAGE パラメーターを使用して、データ型を指定できます。

CHAR-USAGE

COBOL では、各国語データ型 `PIC N` を UTF-16 または DBCS データ用に使用できます。この設定は `NSYMBOL` コンパイラー・オプションによって制御されます。データが適切に扱われるようにするには、アシスタントの **`CHAR-USAGE`** パラメーターを `NSYMBOL` コンパイラー・オプションと同じ値に設定する必要があります。 UTF-16 を使用する場合には、通常、これは `CHAR-USAGE=NATIONAL` に設定されます。

同じコピーブック内で UTF-16 と DBCS のデータを含む各国語データ型を混合する必要がある場合は、個々のフィールドで `USAGE NATIONAL` または `USAGE DISPLAY-1` 修飾子を使用できます。

注: DFHLS2WS、DFHLS2SC、および DFHLS2JS は COBOL の `GROUP USAGE NATIONAL` 節をサポートしません。

アプリケーションからの XML の照会

XML のフラグメントを照会してからアプリケーション・データに変換するアプリケーション・プログラムを作成することができます。

このタスクについて

アプリケーションでさまざまなタイプの XML を処理する場合は、XML を照会して、アプリケーション・データへの変換に使用する XMLTRANSFORM リソースを判別することもできます。 このコマンドは、XML に <xsd:any> エレメントが含まれている場合にも役立つことがあります。

手順

1. アプリケーション・プログラムで XML を照会するには、以下のような **`TRANSFORM XMLTODATA`** API コマンドを使用します。

```
EXEC CICS TRANSFORM XMLTODATA
CHANNEL('MyChannelName')
XMLCONTAINER('SourceContainerName')
ELEMNAME(elementName) ELEMNAMELEN(elementNameLength)
```

   チャネルの名前と、XML を保持するコンテナーの名前を指定する必要があります。 XML を照会するために XMLTRANSFORM リソースを指定する必要はありません。 上記の例では、最初の XML エレメントの名前とその XML エレメントの長さが照会されます。 最初の XML エレメントのタイプ、タイプの長さ、およびタイプの名前空間を照会することもできます。

2. オプション: アプリケーションで XML エレメントの名前空間が必要な場合は、CICS が ELEMNS 値を書き込めるデータ域を提供してください。

3. オプション: XML を照会した後は、XML をアプリケーション・データに変換するために使用する XMLTRANSFORM リソースを判別するアプリケーション・ロジックを作成できます。
4. CICS でアプリケーション・プログラムをインストールします。

タスクの結果

CICS は指定されたコンテナーを読み取り、XML エレメントについての情報をアプリケーション・プログラムに返します。

データ型による XML の取り扱い

XML スキーマにグローバル・データ型が含まれていて、そのうちの 1 つ以上が XML で参照される場合は、これらのグローバル・データ型をサポートするメタデータを生成した後、アプリケーション・プログラムで XML を解析または変換することができます。

始める前に

適切なメタデータを生成するには、TYPES=ALL パラメーターを指定して DFHSC2LS を実行する必要があります。

このタスクについて

手順

1. アプリケーション・データをデータ型により XML に変換するには、以下のような **TRANSFORM DATATOXML** コマンドを使用します。

```
EXEC CICS TRANSFORM DATATOXML
XMLTRANSFORM('
MyXmlTransformName
')
CHANNEL('
MyChannelName
')
DATCONTAINER('
SourceContainerName
')
XMLCONTAINER('
TargetContainerName
')
ELEMNAME(
elementName
) ELEMNAMELEN(
elementNameLength
)
ELEMNS(
elementNamespace
) ELEMNSLEN(
elementNamespaceLength
)
TYPENAME(
typeName
) TYPENAMELEN(
typeNameLen
)
TYPENS(
```

```
typeNamespace
) TYPENSLEN(
typeNamespaceLen
)
```

MyXmlTransformName は、XML バインディングおよびスキーマを指定する XMLTRANSFORM リソースの 32 文字の名前です。*MyChannelName* は、入力および出力コンテナーのあるチャネルの 16 文字の名前です。*SourceContainerName* は、アプリケーション・データを保持する入力コンテナーの 16 文字の名前であり、*TargetContainerName* は、CICS が XML で追加する出力コンテナーの 16 文字の名前です。 XML エレメント名、名前空間、データ型、および変換用のデータ型の名前空間を指定する必要もあります。

2. データ型により XML を解析するには、**TRANSFORM XMLTODATA** コマンドを使用します。 このコマンドで指定するオプションは XML に xsi:type 属性があるかどうかによって異なります。

   - XML で xsi:type 属性が使用されている場合、アプリケーション・プログラムで以下のコマンドを指定します。

     ```
     EXEC CICS TRANSFORM XMLTODATA
     XMLTRANSFORM('
     MyXmlTransformName
     ')
     CHANNEL('
     MyChannelName
     ')
     XMLCONTAINER('
     SourceContainerName
     ')
     DATCONTAINER('
     TargetContainerName
     ')
     ELEMNAME(elementName) ELEMNAMELEN(elementNameLength)
     TYPENAME(typeName) TYPENAMELEN(typeNameLength)
     ```

     アプリケーションでデータ型の名前空間も必要とされる場合、API コマンドに TYPENS オプションを追加します。 データ型は xsi:type 属性から TYPENAME に返されます。

   - XML で xsi:type 属性が使用されていない場合、アプリケーション・プログラムでグローバル・データ型のローカル名および名前空間を指定することができます。 アプリケーション・プログラムで以下のコマンドを使用してください。

     ```
     EXEC CICS TRANSFORM XMLTODATA
     XMLTRANSFORM('
     MyXmlTransformName
     ')
     CHANNEL('
     MyChannelName
     ')
     XMLCONTAINER('
     SourceContainerName
     ')
     DATCONTAINER('
     TargetContainerName
     ')
     TYPENAME(
     typeName
     ) TYPENAMELEN(
     typeNameLength
     ```

```
)
TYPENS(
TypeNamespace
) TYPENSLEN(
typeNamespaceLen
)
```

アプリケーション・プログラムでは、名前および名前空間ではなく型名および名前空間を指定する必要があります。 このコマンドは、アプリケーションが CICS に使用するデータ型を通知することを示しています。 CICS が XML を生成している場合は、常に xsi:type 属性が追加されます。

3. CICS でアプリケーション・プログラムをインストールします。

次のタスク

アプリケーション・プログラムが XML を予期したとおりに生成および解析するかどうかをテストします。

<xsd:any> データ型の処理

1 つ以上の <xsd:any> データ型が含まれる XML スキーマを処理する場合は、XML 支援機能でデータ型を 1 対の CICS コンテナーにマップすることができます。 コンテナー内の XML を解析するアプリケーション・プログラムを作成できます。

始める前に

マッピング・レベル 2.1 以上で DFHSC2LS または DFHWS2LS を使用して XML スキーマをマップする必要があります。

このタスクについて

CICS でデータを XML に変換すると、<xsd:any> データ型と関連付けられた XML が、最初のコンテナーと、2 番目のコンテナーの有効範囲内にある名前空間接頭部宣言に格納されます。

手順

1. この XML データを解析するには、アプリケーション・プログラムで **TRANSFORM XMLTODATA** コマンドを使用します。

```
EXEC CICS TRANSFORM XMLTODATA
XMLTRANSFORM('
MyXmlTransformName
')
CHANNEL('
MyChannelName
')
XMLCONTAINER('
SourceContainerName
')
DATCONTAINER('
TargetContainerName
')
NSCONTAINER('
NamespacesContainerName
')
ELEMNAME(elementName) ELEMNAMELEN(elementNameLength)
```

MyXmlTransformName は、XML バインディングおよびスキーマを指定する XMLTRANSFORM リソースの 32 文字の名前です。*MyChannelName* は、入力および出力コンテナーのあるチャネルの 16 文字の名前です。*SourceContainerName* は、XML を保持する入力コンテナーの 16 文字の名前であり、*TargetContainerName* は、CICS がアプリケーション・データで追加する出力コンテナーの 16 文字の名前です。*NamespacesContainerName* は、 CICS が名前空間接頭部宣言で追加するコンテナーの 16 文字の名前です。ELEMNAME および ELEMNAMELEN オプションの初期値を提供してください。 CICS は XML エレメントと、ELEMNAME および ELEMNAMELEN オプション内にあるその長さを返します。

2. CICS でアプリケーション・プログラムをインストールします。

次のタスク

アプリケーション・プログラムが XML を正常に解析するかどうかをテストします。

XML 変換の妥当性検査

アプリケーション・データをマップするために CICS XML 支援機能を使用する場合、実行時に行う変換が妥当性検査されるように指定して、変換が XML バインディングに含まれるスキーマに準拠することを保証できます。 XML からバイナリー・データまたはバイナリー・データから XML への変換に対して、妥当性検査を実行することができます。

始める前に

CICS アプリケーションの開発およびテストの間に完全な妥当性検査を行うと、XML の問題を検出するのに役立ちます。 ただし、 XML の完全な妥当性検査ではかなりのオーバーヘッドが発生するため、完全にテストされた実動アプリケーションで XML を妥当性検査することはお勧めできません。

CICS では、スキーマに対して XML を妥当性検査するために Java プログラムが使用されます。 したがって、妥当性検査を実行するには、CICS 領域で Java サポートが有効になっている必要があります。

手順

1. CICS 領域で JVM サーバーをセットアップします。 OSGi フレームワークまたは Axis2 で XML バリデーター・クラスを実行できますが、Liberty プロファイルでは実行できません。 CICS には、OSGi フレームワークを使用する JVM サーバーを素早くセットアップするためのサンプルが備わっています。
   a. グループ DFH$OSGI にサンプル JVM サーバー DFHJVMS をインストールするか、独自の JVM サーバーを作成します。 詳しくは、Setting up a JVM serverを参照してください。
   b. 独自の JVM サーバーを作成した場合は、グループ DFHPIVAL 内の DFHPIVAL プログラム定義を変更して、JVMSERVER リソースの名前を参照するようにします。 DFHPIVAL 定義はロックされておらず、編集可能です。 デフォルトでは、この定義は DFHJVMS を参照します。

2. XML バインディングとスキーマが z/OS UNIX 上の同じ場所にあることを確認します。 XMLTRANSFORM リソースで、これらのファイルを CICS に定義します。 **INQUIRE XMLTRANSFORM** コマンドを使用すると、各ファイルの場所を確認することができます。
3. アプリケーションの妥当性検査をオンにします。 CICS Explorer で XMLTRANSFORM リソースを開いて、属性リスト内で妥当性検査の状況フィールドを編集します。あるいは、CEMT または SPI を使用することもできます。

タスクの結果

システム・ログを確認して、XML 変換が有効であるかどうかを調べます。 メッセージ DFHML0508 は、XML が正常に妥当性検査されたことを意味します。メッセージ DFHML0507 は、妥当性検査が失敗したことを意味します。

次のタスク

アプリケーション用の XML 妥当性検査の必要がなくなったら、XMLTRANSFORM リソースを更新してそれをオフにします。

言語構造からのマッピングの生成

XML をアプリケーション・データから作成したりその逆を行ったりする場合には、マッピングを作成して、CICS が実行時にデータおよび XML を変換する方法を記述する必要があります。 これは、どのアプリケーション・データ・レコードからでも開始できます。例えば、COMMAREA、VSAM ファイル、一時ストレージ・キュー、または Db2® レコードから開始できます。

始める前に

マッピングを作成する前に、以下の前提条件が完了していることを確認する必要があります。

- 区分データ・セット内に、アプリケーション・レコードを記述する言語構造が必要です。 言語構造は、COBOL、PL/I、C、および C++ など、CICS XML 支援機能でサポートされている高水準言語のいずれでも記述できます。 このマッピングを Atom フィードで使用する場合で、(作成者名など) Atom エントリーのメタデータを提供するためにアプリケーション・レコードのいずれかのフィールドを使用する場合、そのフィールドが言語構造内でネストされていないことを確認してください。Atom エントリーの内容を提供するフィールド内では、ネストされているフィールドの構造を使用することができます。
- DFHLS2SC が実行されて z/OS UNIX を使用するユーザー ID を構成する必要があります。
- このユーザー ID には、言語構造にアクセスするための読み取り権限と、z/OS UNIX の適切なディレクトリーに出力するための書き込み権限が必要です。
- このユーザー ID で Java を実行するためには、この ID に十分なストレージを割り振る必要があります。 Java のバージョンは、サポートされているものならどれでも使用できます。デフォルトで、DFHLS2SC は **JAVADIR** パラメーターに指定されているバージョンの Java を使用します。

このタスクについて

CICS XML 支援機能を使用して、アプリケーション・レコードのデータ・マッピングを作成します。 CICS XML 支援機能でサポートされている各高水準言語では、いくつかのタイプのデータが制限されているかサポートされていません。 CICS XML 支援機能では、言語構造内で識別された非サポートのアイテムに関して、エラー・メッセージが出されます。 CICS XML 支援機能に関する参照情報では、各高水準言語に適用される制限がリストされています。

手順

1. DFHLS2SC バッチ・ジョブを実行します。 DFHLS2SC には、特定のコード・ページや名前空間の選択など、要件を満たすために選択するオプション・パラメーターが用意されています。 少なくとも、以下のパラメーターを使用します。
   a. 言語構造の高水準言語を **LANG** パラメーターで指定します。
   b. データ・マッピングをバンドルでデプロイする場合、バンドル・リソース名を **BUNDLE** パラメーターで指定します。 Atom フィードの XML バインディングを作成している場合は、このパラメーターを指定しないでください。
   c. マッピング・レベルを **MAPPING-LEVEL** パラメーターで指定します。 Atom フィードの XML バインディングを作成している場合は、マッピング・レベル 3.0 以上を使用する必要があります。 他の状況ではどのマッピング・レベルでも使用できますが、最も高機能のマッピング・オプションを選択するには、最新のマッピング・レベルを使用します。
   d. オプション: Atom フィードの XML バインディングを作成していて、アプリケーション・データ・レコードに CICS ABSTIME 形式のタイム・スタンプが含まれている場合は、オプション・パラメーター **DATETIME=PACKED15** を指定してこれらのフィールドをタイム・スタンプとしてマップしてください。
   e. **PDSMEM** パラメーターと **PDSCP** パラメーターで、アプリケーション・レコードを記述する言語構造の場所とコード・ページを指定します。
   f. スキーマ・ファイルの名前と場所を **SCHEMA** パラメーターで指定します。 ファイル拡張子は `.xsd` です。 バンドルを作成している場合は場所を指定しないでください。 DFHLS2SC は XML スキーマを作成しますが、ファイルがまだ存在していない場合、ディレクトリー構造は作成しません。
   g. XML バインディングの名前と場所を **XSDBIND** パラメーターで指定します。 ファイル拡張子は `.xsdbind` です。バンドルを作成している場合は場所を指定しないでください。 DFHLS2SC は XML バインディングを作成しますが、ファイルがまだ存在していない場合、ディレクトリー構造は作成しません。

      ヒント: XML バインディングとスキーマを同じディレクトリー構造に格納して、妥当性検査を有効にします。 妥当性検査は、アプリケーションを開発またはテスト環境でテストする場合に役立ちます。バンドルを作成している場合、CICS では同じディレクトリーにファイルが配置されます。

   BUNDLE パラメーターを指定した場合、バッチ・ジョブでは z/OS UNIX 上にバンドル・ディレクトリー構造が作成されます。 バンドル・ディレクトリーには、バンドル・マニフェストが含まれる META-INF サブディレクトリーがあり

ます。 バッチ・ジョブは、**SCHEMA** パラメーターと **XSDBIND** パラメーターで指定したファイル名を使用して、XML スキーマと XML バインディングもバンドル・ディレクトリー内に作成します。 **BUNDLE** パラメーターを指定しない場合、バッチ・ジョブにより、XML スキーマと XML バインディングのみが、指定された場所に作成されます。

2. BUNDLE リソース、またはこの XML バインディングを指定する ATOMSERVICE リソースをインストールします。 BUNDLE リソースおよび ATOMSERVICE リソースにより、XML スキーマとバインディング・ファイルの場所を定義する XMLTRANSFORM リソースが動的に作成されます。

タスクの結果

言語構造からマッピングを生成する場合は、1 種類の XML の変換のみが可能です。

例

以下の例は、パラメーターの最小セットが指定された DFHLS2SC を示しています。

```
//LS2SC JOB 'accounting information',name,MSGCLASS=A
// SET QT=''''
//JAVAPROG EXEC DFHLS2SC,
// TMPFILE=&QT.&SYSUID.&QT
//INPUT.SYSUT1 DD *
LANG=COBOL
BUNDLE=/u/exampleapp/bundle/test1
LOGFILE=/u/exampleapp/xsdbind/example.log
MAPPING-LEVEL=3.0
PDSLIB=//CICSHLQ.SDFHSAMP
PDSMEM=CPYBK2
XSDBIND=example.xsdbind
SCHEMA=example.xsd
/*
```

次のタスク

アプリケーション・データを XML に変換したりその逆の変換を行ったりするためのアプリケーション・プログラムを作成します。 両方の変換に同じマッピングを使用できます。 Atom フィードの XML バインディングの作成を完了した後は、Atom フィードをセットアップするステップに進んでください。

XML スキーマからのマッピングの生成

既存の XML スキーマに準拠する XML からアプリケーション・データを作成したりその逆を行ったりする場合には、マッピングを作成して、CICS が実行時にデータを変換する方法を記述する必要があります。これは、XML スキーマまたは WSDL 文書から開始できます。

始める前に

有効な XML スキーマまたは WSDL 文書が必要です。 マッピングを作成する前に、以下の前提条件が完了していることを確認する必要があります。

- 有効な XML スキーマまたは WSDL 文書が必要です。

- UNIX システム・サービスを使用するように、DFHSC2LS を実行するユーザー ID を構成する必要があります。
- このユーザー ID には、XML スキーマまたは WSDL 文書にアクセスするための読み取り権限と、z/OS UNIX の適切なディレクトリーに出力するための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。Java のバージョンは、サポートされているものならどれでも使用できます。デフォルトで、DFHWS2LS は **JAVADIR** パラメーターに指定されているバージョンの Java を使用します。

このタスクについて

CICS XML 支援機能を使用して、XML スキーマのデータ・マッピングを作成します。

手順

1. DFHSC2LS バッチ・ジョブを実行します。 DFHSC2LS には、特定のコード・ページの選択や可変長文字データの処理方法の指定など、要件を満たすために選択できるオプション・パラメーターが用意されています。 少なくとも、以下のパラメーターを使用します。

   a. **WSDL** パラメーターまたは **SCHEMA** パラメーターで、入力ファイルの場所を指定します。 WSDL 文書または XML スキーマを使用できます。 入力ファイルにインターネット上にある他のスキーマまたは文書への参照が含まれていて、システムがプロキシー・サーバーを使用している場合は、そのプロキシー・サーバーのドメイン名か IP アドレス、およびポート番号を指定します。

   b. 生成する高水準言語を **LANG** パラメーターで指定します。 XML 支援機能では、COBOL、C、C++、および PL/I 言語がサポートされています。

   c. データ・マッピングをバンドル内に配置する場合は、**BUNDLE** パラメーターでバンドルの名前と場所を指定します。

   XML 支援機能により、サポートされる変換のライブラリーが XML バインディング内に作成されます。 入力ファイル内のグローバル・エレメントごとに個別の変換が作成されます。

   BUNDLE パラメーターを指定した場合、バッチ・ジョブでは z/OS UNIX 上にバンドル・ディレクトリー構造が作成されます。バンドル・ディレクトリーには、バンドル・マニフェストが含まれる META-INF サブディレクトリーがあります。 バッチ・ジョブではまた、バンドル・ディレクトリー内に XML バインディングが作成され、指定された場所に言語構造が保管されます。 XML 支援機能では、入力ファイルのコピーもバンドル・ディレクトリー内に保管されます。 **BUNDLE** パラメーターを指定しない場合、バッチ・ジョブでは指定された場所に言語構造と XML バインディングのみが作成されます。
2. BUNDLE リソースをインストールします。 BUNDLE リソースによって XMLTRANSFORM リソースが動的に作成されます。このリソースにより、XML スキーマまたは WSDL 文書、XML バインディング、および言語構造のそれぞれの場所が定義されます。

タスクの結果

XML スキーマからマッピングを生成する場合、CICS ではスキーマに存在する各グローバル・エレメント用の言語構造が生成されます。

例

以下の例は、パラメーターの最小セットが指定された DFHSC2LS を示しています。

```
//SC2LS JOB 'accounting information',name,MSGCLASS=A
// SET QT=''''
//JAVAPROG EXEC DFHSC2LS,
// TMPFILE=&QT.&SYSUID.&QT
//INPUT.SYSUT1 DD *
LANG=COBOL
BUNDLE=/u/exampleapp/bundle/test1
LOGFILE=/u/exampleapp/xsdbind/example.log
MAPPING-LEVEL=3.0
PDSLIB=//CICSHLQ.SDFHSAMP
PDSMEM=CPYBK2
XSDBIND=example.xsdbind
SCHEMA=example.xsd
/*
```

次のタスク

アプリケーション・データを XML に変換したりその逆の変換を行ったりするためのアプリケーション・プログラムを作成します。両方の変換に同じマッピングを使用できます。

アプリケーション・データの XML への変換

アプリケーション・プログラムを書き込み、アプリケーション・データを XML に変換できます。

始める前に

XML バインディングと XML スキーマを定義する、有効な XMLTRANSFORM リソースが必要です。

このタスクについて

XML 支援機能は、XML バインディング内のマッピングを生成します。言語構造から始めて DFHLS2SC を使用した場合、可能な XML への変換は 1 種類だけです。XML スキーマから始めた場合は、XML から言語構造への変換は複数の種類が可能であるため、生成する XML エレメントをアプリケーションで選択する必要があります。

手順

1. アプリケーション・プログラムはチャネルを作成し、言語構造に関連付けられたデータをそのチャネルのビット・モード・コンテナーに格納する必要があります。
2. **TRANSFORM DATATOXML** API コマンドを使用して、データを XML に変換します。

```
EXEC CICS TRANSFORM DATATOXML
XMLTRANSFORM('
MyXmlTransformName
')
CHANNEL('
MyChannelName
')
DATCONTAINER('
SourceContainerName
')
XMLCONTAINER('
TargetContainerName
')
```

XMLTRANSFORM リソースでは、サポートされている言語構造からの変換が 1 種類だけであるため、変換タイプをコマンドで指定する必要はありません。 複数の種類の変換が可能である場合は、以下のオプションをアプリケーション・プログラムに追加してください。

```
ELEMNAME(elementName) ELEMNAMELEN(elementNameLength)
```

これらの追加オプションは、アプリケーション・データが変換されて出力コンテナーに配置される XML エレメントを示します。

3. アプリケーション・プログラムをインストールします。

タスクの結果

アプリケーションが **TRANSFORM DATATOXML** コマンドを実行すると、CICS は XMLTRANSFORM リソースをチェックして XML バインディングのマッピングを検索し、チャネル上のコンテナーを使用してアプリケーション・バイナリー・データを XML に変換します。 XML は、アプリケーションが XMLCONTAINER オプションで指定したコンテナーに格納されます。 この XML は、XMLTRANSFORM リソースで定義されている XML スキーマに準拠しています。

次のタスク

同じマッピングを使用して XML をアプリケーション・データに変換することもできます。 詳細については、『アプリケーション・データへの XML の変換』を参照してください。

アプリケーション・データへの XML の変換

XML をアプリケーション・データに変換するためのアプリケーション・プログラムを作成できます。 変換前に XML を照会することもできます。

始める前に

XML バインディングと XML スキーマを定義する、有効な XMLTRANSFORM リソースが必要です。

このタスクについて

CICS の XML 支援機能は、XML バインディング内のマッピングを生成します。 言語構造から始めて DFHLS2SC を使用した場合、可能な XML からの変換は 1 種類だけです。 XML スキーマから始めた場合は、XML から言語構造への変換は複

数の種類が可能であるため、出力として使用する言語構造をアプリケーションで選択する必要があります。

手順

1. チャネルを作成し、XML をそのチャネルでテキスト・モード・コンテナーに格納します。 XML がアプリケーションによって BIT モード・コンテナーに格納されると、 CICS はテキスト・データのエンコードを判別しようとしますが、コンテナーの処理に時間がかかり、エンコードが正しくならない場合があります。
2. **TRANSFORM XMLTODATA** API コマンドを使用します。 可能な XML の変換が 1 種類だけの場合は、以下のコマンドを使用することができます。

   ```
   EXEC CICS TRANSFORM XMLTODATA
   XMLTRANSFORM('
   MyXmlTransformName
   ')
   CHANNEL('
   MyChannelName
   ')
   XMLCONTAINER('
   SourceContainerName
   ')
   DATCONTAINER('
   TargetContainerName
   ')
   ```

   複数の種類の変換が可能な場合は、以下の追加のオプションを使用してください。

   ```
   EXEC CICS TRANSFORM XMLTODATA
   XMLTRANSFORM('
   MyXmlTransformName
   ')
   CHANNEL('
   MyChannelName
   ')
   XMLCONTAINER('
   SourceContainerName
   ')
   DATCONTAINER('
   TargetContainerName
   ')
   ELEMNAME(
   elementName
   ) ELEMNAMELEN(
   elementNameLength
   )
   ```

   2 番目の例では、ELEMNAME オプション内で見つかったエレメントの名前が返されます。 アプリケーションでそのエレメントの名前を使用して、ターゲット・コンテナーの内容を解釈するために使用する言語構造のライブラリーを判別できます。
3. アプリケーション・プログラムをインストールします。

タスクの結果

アプリケーションによって **TRANSFORM XMLTODATA** コマンドが実行されると、CICS は XMLTRANSFORM リソース内の詳細を使用して、 XML をチャネル上のコンテナーを使用するアプリケーション・バイナリー・データに変換します。

例

次のタスク

同じマッピングを使用してアプリケーション・データを XML に変換することもできます。 詳細については、 449 ページの『アプリケーション・データの XML への変換』を参照してください。

アプリケーション・データおよび JSON のマッピングと変換

アプリケーション・プログラムを作成して、アプリケーションのバイナリー・データを JavaScript Object Notation (JSON) に変換したり逆の変換を行ったりすることができます。CICS は、多数の高水準言語をサポートし、ランタイム処理中のデータの変換方法をマップするための JSON 支援機能を提供します。 CICS は Web サービス・サポートの一部として、同じテクノロジーを使用してアプリケーション・データを JSON メッセージにマップします。

始める前に

JSON 支援機能を実行するには Java がインストールされている必要があります。変換は、CICS を使用して内部で行うか、JVM サーバーを使用して行うことができます。変換に Java を使用する場合、アプリケーション・データと JSON を変換するためには、Axis2 JVM サーバーがインストールされている必要があります。詳細については、Axis2 用の JVM サーバーの構成を参照してください。

このタスクについて

このアプローチを使用してアプリケーション・データから JSON への (またはその逆の) 変換を行う利点は、JSON パーサーが提供する機能を超える処理が CICS によって行われることです。 CICS は、JSON を解釈して、アプリケーション・データのレコード・ベース変換を行うことができます。 したがって、このアプローチを使用することで、JSON を処理するアプリケーションをより簡単かつ迅速に作成できます。

付属のユーティリティーである CICS の JSON 支援機能は、アプリケーションのバイナリー・データを JSON に変換したり、JSON をアプリケーションのバイナリー・データに変換したりするために必要なマッピング成果物の作成を支援します。JSON 支援機能は、成果物をバンドル・ディレクトリーに作成します。

手順

1. JSON 支援機能を使用して、バンドルを作成します。 このバンドルには、データ変換に必要なマッピング成果物が含まれます。
2. CICS でバンドルをインストールして、マッピングを使用可能にします。
3. データ変換を処理するようアプリケーション・プログラムを作成または更新します。 次の 2 つのオプションがあります。
   - アプリケーション・プログラムで **TRANSFORM DATATOJSON** および **TRANSFORM JSONTODATA** API コマンドを使用します。これが推奨される方法です。
   - **LINK PROGRAM** API コマンドを使用して、CICS 提供のリンク可能インターフェース DFHJSON にリンクします。

そのアプリケーションはチャネル・ベースのインターフェースを使用する必要があります。

4. アプリケーションを実行して、変換が意図したとおりに行われるかどうかをテストします。

タスクの結果

アプリケーション・データが JSON に変換されるか、JSON がアプリケーション・データに変換されます。

次のタスク

以下のトピックで、ステップ 1 (452 ページ) から 4 について、詳しく説明されています。

CICS JSON アシスタント

CICS JSON アシスタントは、JSON とアプリケーション・データとの間の変換を行うために使用する、高水準言語構造と JSON スキーマ間のマッピングを生成できるよう支援するバッチ・ユーティリティーのセットです。この支援機能により、JSON 処理を実行するアプリケーションを最小限のプログラミングで迅速にデプロイすることが可能になります。

CICS 用の JSON アシスタントを使用すると、JSON の解析または構成のために作成する必要のあるコードの量を減らすことができます。CICS では、JSON フラグメントとアプリケーション・プログラムのデータ構造との間でデータが相互に変換されます。

JSON 支援機能では、単純な言語構造からの JSON スキーマの作成や既存の XML スキーマからの言語構造の作成が可能であり、COBOL、C/C++、および PL/I をサポートしています。また、 CICS が実行時に XML データをバイナリー・アプリケーション・データに自動的に変換する (または逆の変換を行う) ために使用するメタデータも生成されます。このメタデータは XML バインディングで定義され、z/OS UNIX 上に保管されます。 XML バインディング用のスキーマは、z/OS UNIX 上の `/usr/lpp/cicsts/cicsts52/schemas/xmltransform/` ディレクトリーにあります。

ユーティリティー・プログラム DFHJS2LS と DFHLS2JS を合わせて、CICS JSON アシスタントといいます。

DFHLS2JS
DFHLS2JS は、高水準言語構造を JSON スキーマにマッピングします。

DFHJS2LS
DFHJS2LS は、JSON スキーマを高水準言語構造にマッピングします。

hlq.XDFHINST ライブラリーには、両方のプログラムを実行するための JCL プロシージャーがあります (*hlq* は CICS インストールの高位修飾子です)。

DFHLS2JS または DFHJS2LS プロシージャーに関連する使用モードは、要件に応じて次のように異なります。

- DFHLS2JS: リンク可能インターフェースに関する高水準言語から JSON スキーマへの変換
- DFHJS2LS: リンク可能インターフェースに関する JSON スキーマから高水準言語への変換
- DFHLS2JS: 要求/応答サービスに関する高水準言語から JSON スキーマへの変換
- DFHJS2LS: 要求/応答サービスに関する JSON スキーマから高水準言語への変換
- DFHJS2LS: RESTful サービスにおける JSON スキーマから高水準言語への変換

以下に示すように、2 つのマッピングは対称的ではありません。

- DFHLS2JS を使用して言語データ構造を処理し、結果として生成される JSON スキーマを DFHJS2LS で処理する場合、最終的なデータ構造と元の構造が同じであると期待することはできません。
- DFHJS2LS を使用して JSON スキーマを処理し、結果として生成された言語構造を DFHLS2JS を使用して処理する場合、最終的な JSON スキーマと元のスキーマが同じであると期待することはできません。
- 場合によっては、DFHJS2LS により、DFHLS2JS でサポートされていない言語構造が生成されます。

DFHLS2JS によって処理される高水準言語構造は、CICS によってサポートされ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コーディングする必要があります。

DFHLS2JS: リンク可能インターフェースに関する高水準言語から JSON スキーマへの変換

カタログ式プロシージャーの DFHLS2JS は、JSON スキーマと JSON バインディング・ファイルを高水準言語構造から生成します。 JSON の構文解析または作成が可能な CICS プログラムを作成する場合は、DFHLS2JS を使用してください。

DFHLS2JS JCL プロシージャーは、データ・セット *HLQ*.XDFHINST にインストールされます。ここで、*HLQ* は、CICS がインストールされている高位修飾子です。

DFHLS2JS 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 (DFHLS2JS) を指定します。

INPUT.SYSUT1 DD
入力を指定します。 入力パラメーターは、入力ストリームで指定します。ただし、データ・セットまたは区分データ・セットのメンバーで定義することもできます。

シンボリック・パラメーター

DFHLS2JS では以下のシンボリック・パラメーターが定義されています。

JAVADIR = *path*
DFHLS2JS 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が /usr/lpp/ に付加されることにより、完全なパス名 /usr/lpp/*path* が形成されます。

通常、このパラメーターは指定しません。 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。

PATHPREF = *prefix*
他のパラメーターで使用される z/OS UNIX ディレクトリー・パスを拡張する、オプションの接頭部を指定します。 デフォルトは空ストリングです。

通常、このパラメーターは指定しません。 デフォルト値は、**PATHPREF** 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。

SERVICE = *value*
このパラメーターは、IBM サポートに指示された場合にのみ使用します。

TMPDIR = *tmpdir*
DFHLS2JS が一時ワークスペースとして使用する z/OS UNIX 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*
DFHLS2JS が一時ワークスペース・ファイルの名前を構成するために使用する接頭部を指定します。

デフォルト値は LS2JS です。

USSDIR = *path*
z/OS UNIX ファイル・システム内の 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/ に付加されて、/usr/lpp/cicsts/*path* の完全なパス名が形成されます。デフォルトを使用する場合は、これを '.' (ピリオド) として指定する必要があります。

通常、このパラメーターは指定しません。 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。

一時ワークスペース

DFHLS2JS により、実行時に以下の 3 つの一時ファイルが作成されます。

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名 (**TMPDIR** および **TMPFILE** が指定されていない場合) は、次のとおりです。

/tmp/LS2JS.in

```
/tmp/LS2JS.out
/tmp/LS2JS.err
```

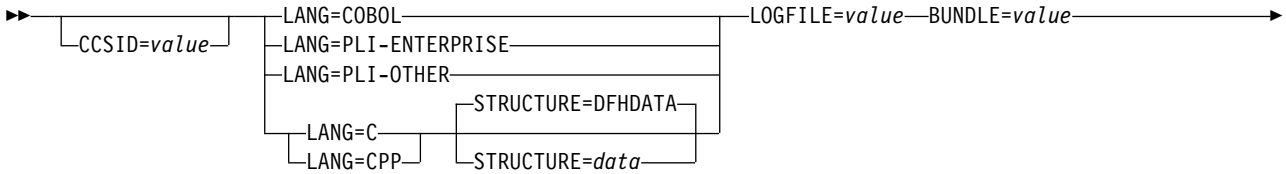
重要: DFHLS2JS は、z/OS UNIX ファイルまたはデータ・セット・メンバーへのアクセスをロックしません。したがって、DFHLS2JS の複数のインスタンスが同時に実行され、同じ一時ワークスペース・ファイルを使用する場合、あるジョブがワークスペース・ファイルを使用しているときに、他のジョブがそれらのファイルを上書きするのを防止することはできないため、予測不能な障害が生じます。

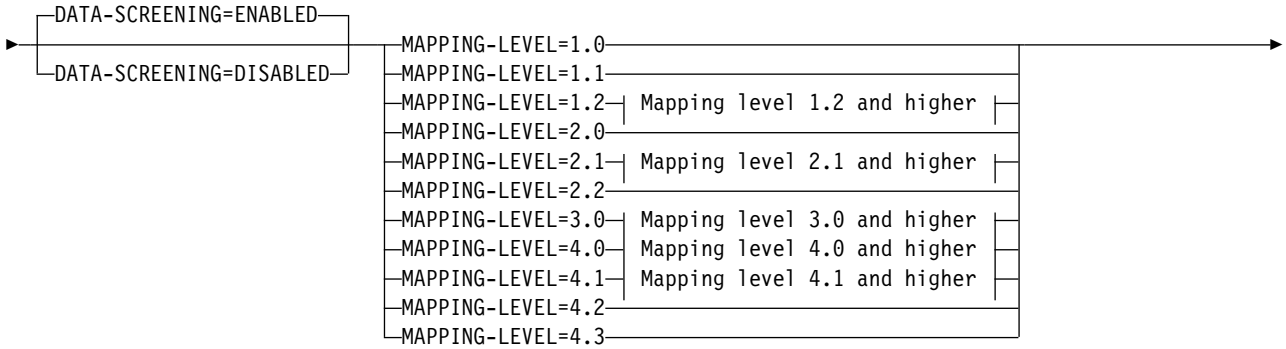
このような状況を回避する命名規則と操作手順を考案してください。例えば、システム・シンボリック・パラメーターの **SYSUID** を使用して、個々のユーザーにとって固有のワークスペース・ファイル名を生成することができます。

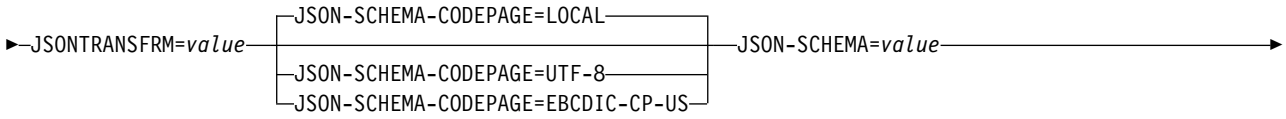
これらの一時ファイルは、ジョブが終了する前に削除されます。

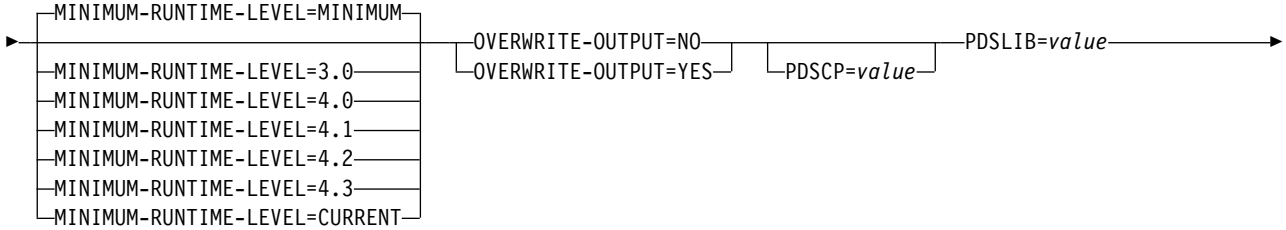
DFHLS2JS の入力パラメーター

以下の構文図は、使用可能な入力パラメーターを示しています。

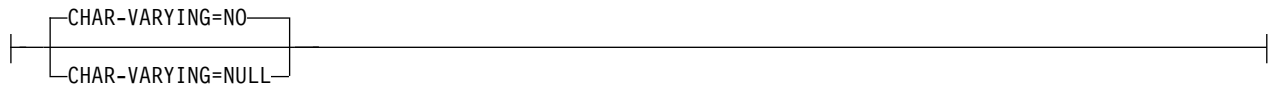






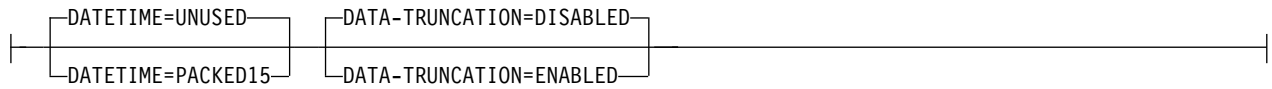


マッピング・レベル **1.2** 以上の場合:

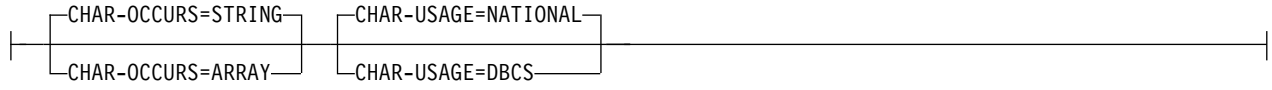


マッピング・レベル **2.1** 以上の場合:

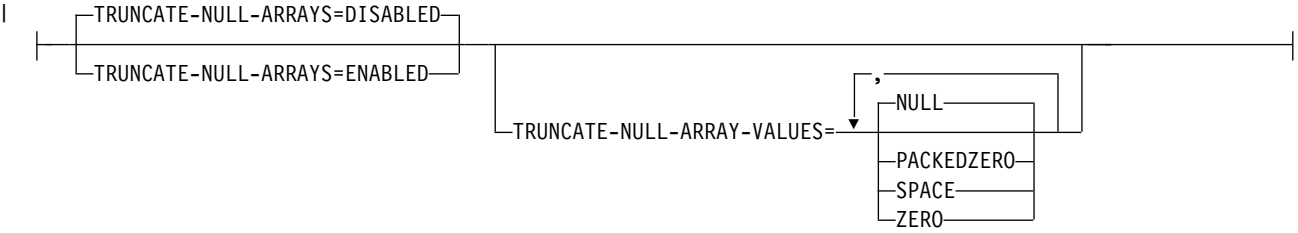
マッピング・レベル **3.0** 以上:



マッピング・レベル **4.0** 以上の場合:



マッピング・レベル **4.1** 以上の場合:



パラメーターの使用法

パラメーターは、以下の規則に準拠していなければなりません。

- 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 (および使用する場合は、その継続文字) は、72 桁を超えて拡張することはできません。73 桁から 80 桁の間はブランクにする必要があります。

- 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。 アスタリスクの前のすべての部分 (スペースを含む) は、パラメーターの一部とみなされます。
- 行の先頭の文字の位置に番号記号 (#) 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。
- 行の末尾の文字の位置にコンマがある場合、これはオプションの行分離文字であり、無視されます。

パラメーターの記述

BUNDLE = *value*

z/OS UNIX 上のバンドル・ディレクトリーのパスと名前を指定します。この値を指定すると、JSON 支援機能によって、SON バインディングを組み込んだバンドルが生成されます。 このパラメーターのパス情報は、**JSONTRANSFRM** パラメーターのパス情報を指定変更します。

ディレクトリー名の代わりにアーカイブ・ファイルを指定することもできます。JSON 支援機能では `.zip` および `.jar` アーカイブがサポートされています。ただし、BUNDLE リソースをインストールする前にアーカイブを圧縮解除する必要があります。

CCSID = *value*

アプリケーション・データ構造に文字データをエンコードするために、実行時に使用される CCSID を指定します。このパラメーターの値は、**LOCALCCSID** システム初期設定パラメーターの値を指定変更します。 *value* は、Java および z/OS 変換サービスによってサポートされている EBCDIC CCSID である必要があります。このパラメーターを指定しない場合、アプリケーション・データ構造は、システム初期設定パラメーターで指定された CCSID を使用してエンコードされます。

このパラメーターは任意のマッピング・レベルで使用することができます。

CHAR-VARYING = { **NO** | **NULL** | **COLLAPSE** | **BINARY** }

マッピング・レベルが 1.2 以上のとき、言語構造内の文字フィールドがマップされる方法を指定します。 COBOL の文字フィールドは、タイプ X のピクチャー節 (`PIC(X) 10` など) です。 C/C++ の文字フィールドは文字配列です。このパラメーターは、Enterprise および他の PL/I 言語構造には適用されません。 以下のオプションを選択できます。

NO
: 文字フィールドは JSON ストリングにマップされ、固定長のフィールドとして処理されます。 データの最大長はフィールドの長さと同じです。NO は、マッピング・レベル 2.0 以前での、COBOL および PL/I の **CHAR-VARYING** パラメーターのデフォルト値です。

NULL
: 文字フィールドは JSON ストリングにマップされ、ヌル終了ストリングとして処理されます。CICS は、JSON スキーマから変換する際に、終了ヌル文字を追加します。文字ストリングの最大長は、言語構造に示される長さより 1 文字少ない長さとして計算されます。 NULL は、C/C++ での **CHAR-VARYING** パラメーターのデフォルト値です。

COLLAPSE

文字フィールドは、JSON ストリングにマップされます。フィールド内の後続空白と埋め込み空白は JSON メッセージに含まれません。例えば、<space>AB<space><space><space>C<space> は AB<space>C になります。マッピング・レベル 2.1 以降の COBOL および PL/I では、**CHAR-VARYING** パラメーターのデフォルト値は COLLAPSE です。

BINARY

文字フィールドは、base64 エンコード・データを含む JSON ストリングにマップされ、固定長のフィールドとして処理されます。**CHAR-VARYING** パラメーターで BINARY 値が使用できるのは、マッピング・レベル 2.1 以降のみです。

CHAR-OCCURS = { **STRING** | **ARRAY** }

マッピング・レベル 4.0 以上の場合に、言語構造内の文字配列をマップする方法を指定します。例えば PIC X OCCURS 20 となります。このパラメーターは COBOL 言語でのみ使用されます。

ARRAY

文字配列は JSON 配列にマップされます。つまり、それぞれの文字が個別の JSON エレメントとしてマップされます。マッピング・レベル 3.0 以前でも、このように動作します。

STRING

文字配列は JSON ストリングにマップされます。つまり、COBOL 配列全体が 1 つの JSON エレメントとしてマップされます。

CHAR-USAGE = { **NATIONAL** | **DBCS** }

COBOL では、各国語データ型 PIC N を UTF-16 または DBCS データ用に使用できます。この設定は NSYMBOL コンパイラー・オプションによって制御されます。データが適切に扱われるようにするには、アシスタントの **CHAR-USAGE** パラメーターを NSYMBOL コンパイラー・オプションと同じ値に設定する必要があります。 UTF-16 を使用する場合には、通常、これは CHAR-USAGE=NATIONAL に設定されます。

DBCS

PIC (*n*) フィールドからのデータは、UTF-16 でエンコードされたデータとして扱われます。

NATIONAL

PIC (*n*) フィールドからのデータは、DBCS でエンコードされたデータとして扱われます。

DATA-SCREENING = { **ENABLED** | **DISABLED** }

アプリケーション提供のデータのエラーを検査するかどうかを指定します。

ENABLED

言語構造と矛盾するアプリケーション提供のランタイム・データは、エラーとして扱われ、メッセージ DFHPI1010 が発行されます。エラー応答がアプリケーションに 返されます。

DISABLED

言語構造と矛盾するアプリケーション提供のランタイム・データの値は、 デフォルト値で置き換えられます。例えば、数値フィールド内の

ゼロは誤った値を置き換えます。メッセージ DFHPI1010 は発行されず、通常の応答がアプリケーションに返されます。この機能を使用して、 初期設定されていない出力フィールドから生成される INVALID_PACKED_DEC および INVALID_ZONED_DEC エラー応答を回避できます。

DATA-TRUNCATION = { **DISABLED** | **ENABLED** }
固定長フィールド構造で可変長データを許容するかどうかを指定します。

DISABLED
データが CICS が予期する固定長より短い場合に、CICS は切り捨てられたデータを拒否してエラー・メッセージを発行します。

ENABLED
データが CICS が予期する固定長より短い場合に、CICS は切り捨てられたデータを許容して、欠落データをヌル値として処理します。

DATETIME = { **UNUSED** | **PACKED15** }
高水準言語構造内の JSON 日時プロパティー (CICS ABSTIME 値を含む) がタイム・スタンプとしてマップされるかどうかを指定します。

PACKED15
すべての JSON 日時プロパティーがタイム・スタンプとしてマップされます。

UNUSED
いかなる JSON 日時プロパティーもタイム・スタンプとしてマップされません。 このマッピングはデフォルトです。

このパラメーターは、マッピング・レベル 3.0 で設定できます。

JSON-SCHEMA = *value*

JSON スキーマが書き込まれるファイルの、完全修飾された z/OS UNIX 名。

JSON-SCHEMA-CODEPAGE = { **LOCAL** | **UTF-8** | **EBCDIC-CP-US** }
JSON スキーマ文書を生成するために使用されるコード・ページを指定します。

LOCAL
ファイル・システムのデフォルトのコード・ページを使用して生成された JSON スキーマを指定します。

UTF-8
UTF-8 コード・ページを使用して生成された JSON スキーマを指定します。

EBCDIC-CP-US
US EBCDIC コード・ページを使用して生成された JSON スキーマを指定します。

JSONTRANSFRM = *value*
このパラメーターは、LINKable モードでは必須ですが、要求/応答モードでは無効です。これは、CICS の **JSONTRANSFRM** バンドル・リソースに使用される名前を指定します。

LANG = **COBOL**|**PLI-ENTERPRISE**|**PLI-OTHER**|**C**|**CPP**
高水準言語構造のプログラミング言語を指定します。

COBOL
COBOL

PLI-ENTERPRISE
Enterprise PL/I

PLI-OTHER
Enterprise PL/I 以外の PL/I のレベル

C C

CPP C++

LOGFILE = *value*
DFHLS2JS 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 z/OS UNIX 名です。 この名前が存在しない場合、DFHLS2JS はファイルを作成しますが、ディレクトリー構造は作成しません。

DFHLS2JS に問題が発生した場合、このパラメーターを使用するよう、IBM のサービス組織から依頼される場合があります。

MAPPING-LEVEL = **{ 1.0 | 1.1 | 1.2 | 2.0 | 2.1 | 2.2 | 3.0 | 4.0 | 4.1 | 4.2 | 4.3 }**
JSON バインディングおよび言語構造の生成時に支援機能で使用するマッピング・レベルを指定します。 マッピング・レベル 3.0 以上を使用する必要があります。

MINIMUM-RUNTIME-LEVEL = **{ MINIMUM | 3.0 | 4.0 | 4.1 | 4.2 | 4.3 | CURRENT }**
JSON バインディングをデプロイすることが可能な、CICS の最小ランタイム環境を指定します。 指定した他のパラメーターと一致しないレベルを選択すると、エラー・メッセージが送信されます。 選択可能なオプションは以下のとおりです。

MINIMUM
選択したパラメーターを前提として、最低限可能な CICS 実行時レベルが自動的に割り振られます。

3.0 CICS JSON 支援機能を使用して、高機能のデータ・マッピングを利用する場合は、ランタイム・レベル 3.0 以上を指定します。

4.0 生成された Web サービス・バインディング・ファイルは、CICS TS 5.2 以降の領域に正常にデプロイされます。この実行時レベルでは、**MAPPING-LEVEL** パラメーターにマッピング・レベル 4.0 以前を使用できます。このレベルでは任意のオプション・パラメーターを使用できます。

4.1 生成された Web サービス・バインディング・ファイルは、CICS TS 5.2 以降の領域に正常にデプロイされます。この実行時レベルでは、**MAPPING-LEVEL** パラメーターには、マッピング・レベル 4.1 以前を使用することができます。

4.2 生成された Web サービス・バインディング・ファイルは、CICS TS V5.4 以降の領域に正常にデプロイされます。この実行時レベルでは、**MAPPING-LEVEL** パラメーターには、マッピング・レベル 4.2 以前を使用することができます。

4.3 生成された Web サービス・バインディング・ファイルは、CICS TS

5.4 以降の領域に正常にデプロイされます。この実行時レベルでは、**MAPPING-LEVEL** パラメーターには、マッピング・レベル 4.3 以前を使用することができます。

CURRENT
生成されたバインディング・ファイルを、ランタイム環境がバインディング・ファイルの生成に使用された領域と同じである CICS 領域にデプロイする場合、このランタイム・レベルを使用します。

OVERWRITE-OUTPUT ={NO | YES}
ファイル・システム上の既存の CICS BUNDLE が上書き可能かどうかを制御します。

NO 既存 BUNDLE は何も置き換えられません。 既存 BUNDLE が検出された場合、DFHLS2JS はエラー・メッセージ DFHPI9689E を発行し強制終了します。

YES どの既存 BUNDLE も置き換えられます。 既存 BUNDLE が検出された場合、メッセージ DFHPI9683W が発行され、ファイルが置き換えられたことを通知します。

PDSCP = *value*
区分データ・セット・メンバーで使用されるコード・ページを指定します。ここで、*value* は CCSID 番号または Java コード・ページ番号です。 このパラメーターを指定しない場合、z/OS UNIX システム・サービス・コード・ページが使用されます。例えば、PDSCP=037 を指定することができます。

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。

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。

PDSMEM = *value*
処理する高水準言語構造を含む区分データ・セット・メンバーの名前を指定します。

STRUCTURE ={DFHDATA | *data*}
C および C++ での、最上位データ構造の名前。デフォルトは DFHDATA です。

TRUNCATE-NULL-ARRAYS = { **DISABLED** | **ENABLED** }
マッピング・レベル 4.1 以上で構造化配列を処理する方法を指定します。この機能を有効にすると、CICS は配列に含まれる空のレコードを認識しようと試みます (空のレコードの認識について詳しくは、TRUNCATE-NULL-ARRAY-VALUES を参照)。空の配列レコードが 5 個連続して検出された場合、その配列は XML/JSON の生成時に最初の空のレコードの箇所で切り捨てられます。この機能は、内容が構造化されている配列に対してのみ有効にされます。単純なプリミティブ・フィールドからなる配列には、切り捨ては適用されません。配列の切り捨てにより、JSON/XML 内のデータ表現が簡潔になりますが、リスクがないわけではありません。5 個の連続したデータ・レコードが、(おそらく、正当な低値を含んでいるために) 初期化されていないストレージとして誤って認識された場合、データ損失が生じるおそれがあります。TRUNCATE-NULL-

ARRAYS が有効にされていて、TRUNCATE-NULL-ARRAY-VALUES が設定されていない場合は、TRUNCATE-NULL-ARRAY-VALUES のデフォルト値が使用されます。

TRUNCATE-NULL-ARRAY-VALUES = { **NULL** | **PACKEDZERO**|**SPACE** | **ZERO** }

マッピング・レベル 4.1 以上で、TRUNCATE-NULL-ARRAYS の処理で空として扱う値を指定します。デフォルトでは、ヌル値 (0x00、または小さい値) が空として扱われます。構造化された配列のレコードに含まれるすべてのストレージ・バイトにヌルが含まれている場合、レコード全体が空であると見なされます。1 つ以上の NULL、PACKEDZERO、SPACE、および ZERO 値をコンマ区切りリストに指定できます。

NULL
ヌル文字 (0x00) を暗黙指定します。

PACKEDZERO
正符号付きパック 10 進数ゼロ (0x0C)、負符号付きパック 10 進数ゼロ (0x0D)、または符号なしパック 10 進数ゼロ (0x0F) を暗黙指定します。

SPACE
SBCS EBCDIC スペース (0x40) を暗黙指定します。

ZERO
符号なしのゾーン 10 進数ゼロ (0xF0) を暗黙指定します。

構造化配列レコードに、選択したバイトの組み合わせとの一致が含まれている場合、レコード全体が空として識別されます。

TRUNCATE-NULL-ARRAY-VALUES に値が定義されている場合、TRUNCATE-NULL-ARRAYS が有効に設定されている必要があります。

```
//LS2JS JOB '
accounting information
',
name,MSGCLASS=A
// SET QT=''''
//JAVAPROG EXEC DFHLS2JS,
// TMPFILE=&QT.&SYSUID.&QT
/INPUT.SYSUT1 DD *
PDSLIB=//CICSHLQ.SDFHSAMP
PDSMEM=CPYBK2
JSON-SCHEMA=example.json
LANG=COBOL
LOGFILE=/u/exampleapp/example.log
MAPPING-LEVEL=4.0
JSONTRANSFRM=EXAMPLE
BUNDLE=/u/exampleapp/bundles/exampleBundle
/*
```

DFHJS2LS: リンク可能インターフェースに関する JSON スキーマから高水準言語への変換

DFHJS2LS カタログ式プロシージャーは JSON スキーマから高水準言語データ構造および JSON バインディングを生成します。 JSON の構文解析または作成が可能な CICS プログラムを作成する場合は、DFHJS2LS を使用してください。 このトピックでは、DFHJS2LS のジョブ制御ステートメント、シンボリック・パラメーター、入力パラメーターおよびそれぞれの説明を記載します。

DFHJS2LS JCL プロシージャーは、データ・セット *HLQ*.XDFHINST にインストールされます。ここで、*HLQ* は、CICS がインストールされている高位修飾子です。

DFHJS2LS 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 (DFHJS2LS) を指定します。

INPUT.SYSUT1 DD
入力を指定します。 入力パラメーターは、入力ストリームで指定します。データ・セットまたは区分データ・セットのメンバーで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、DFHJS2LS で定義されます。

JAVADIR = *path*
DFHJS2LS 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が /usr/lpp/ に付加されることにより、完全なパス名 /usr/lpp/*path* が形成されます。

通常、このパラメーターは指定しません。 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。

PATHPREF = *prefix*
他のパラメーターで使用される z/OS UNIX ディレクトリー・パスを拡張する、オプションの接頭部を指定します。 デフォルトは空ストリングです。

通常、このパラメーターは指定しません。 デフォルト値は、**PATHPREF** 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。

SERVICE = *value*
このパラメーターは、IBM サポートに指示された場合にのみ使用します。

TMPDIR = *tmpdir*
DFHJS2LS が一時ワークスペースとして使用する z/OS UNIX 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・ファイルの名前を構成するときに DFHJS2LS によって使用される接頭部を指定します。

デフォルト値は JS2LS です。

USSDIR = *path*
UNIX システム・サービスのファイル・システム内の 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/ に付加されて、/usr/lpp/cicsts/*path* という完全なパス名が生成されます。デフォルトを使用する場合は、これを '.' (ピリオド) として指定する必要があります。

通常、このパラメーターは指定しません。 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。

一時ワークスペース

DFHJS2LS 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名 (**TMPDIR** および **TMPFILE** が指定されていない場合) は、次のとおりです。

/tmp/JS2LS.in

/tmp/JS2LS.out

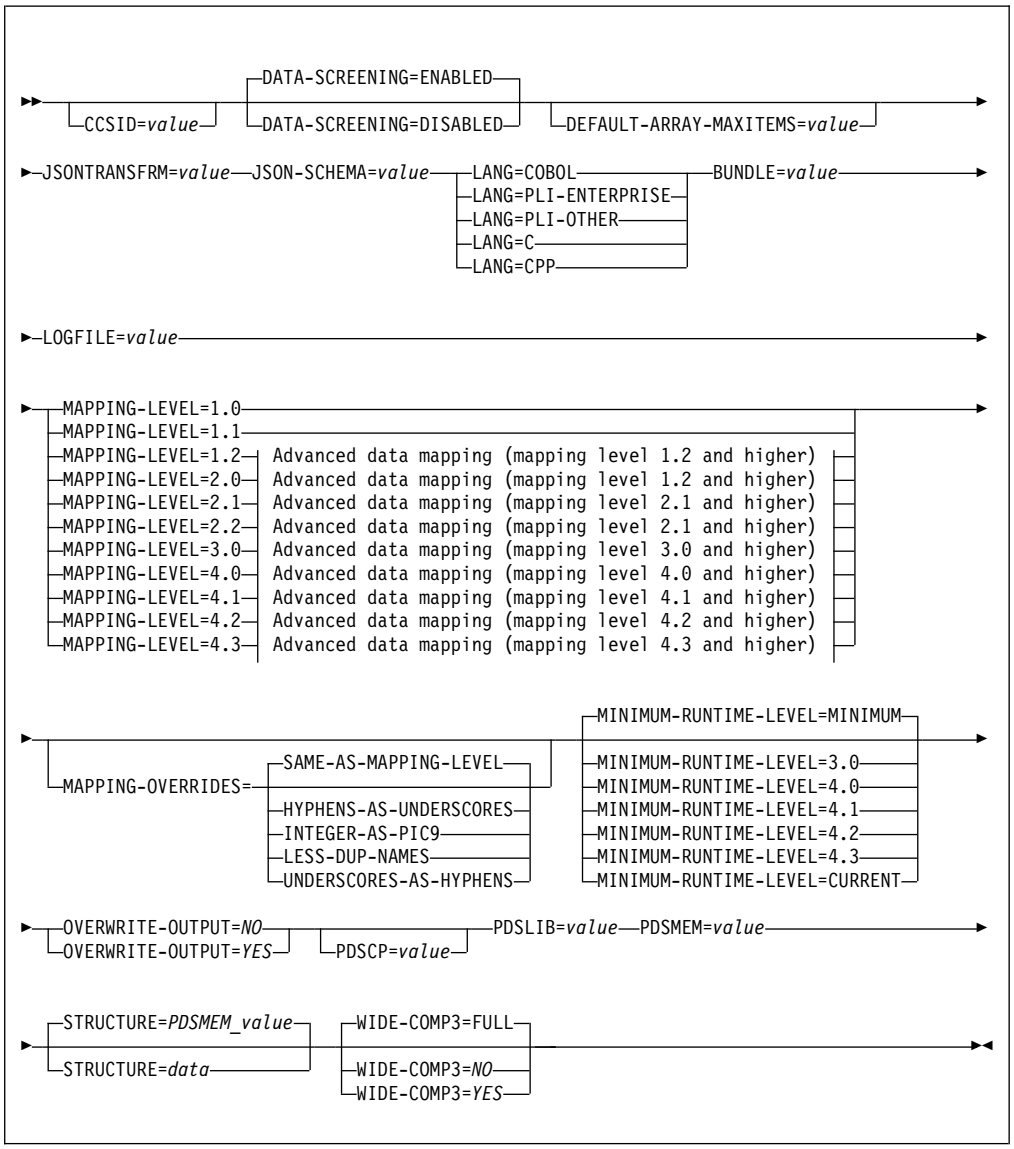
/tmp/JS2LS.err

重要: DFHJS2LS は、z/OS UNIX ファイルまたはデータ・セット・メンバーへのアクセスをロックしません。したがって、DFHJS2LS の複数のインスタンスが同時に実行され、同じ一時ワークスペース・ファイルを使用する場合、あるジョブがワークスペース・ファイルを使用しているときに、他のジョブがそれらのファイルを上書きするのを防止することができないため、予測不能な障害が生じます。

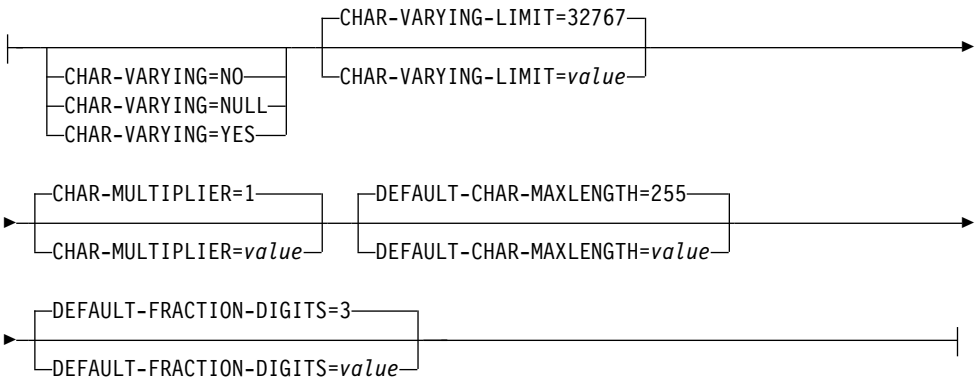
そのため、このような状況を回避する命名規則と操作手順を考案することをお勧めします。例えば、システム・シンボリック・パラメーターの **SYSUID** を使用して、個々のユーザーにとって固有のワークスペース・ファイル名を生成することができます。

これらの一時ファイルは、ジョブが終了する前に削除されます。

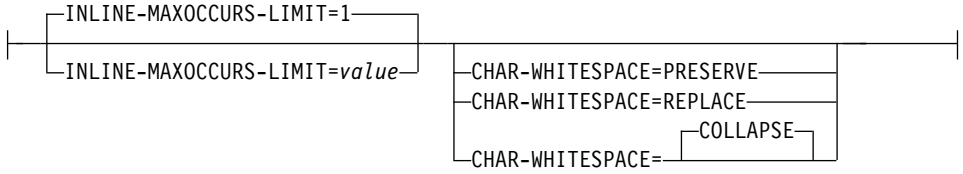
DFHJS2LS のパラメーターの入力



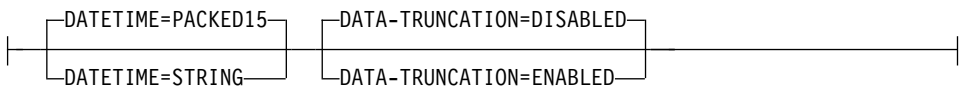
拡張データ・マッピング (マッピング・レベル **1.2** 以上):



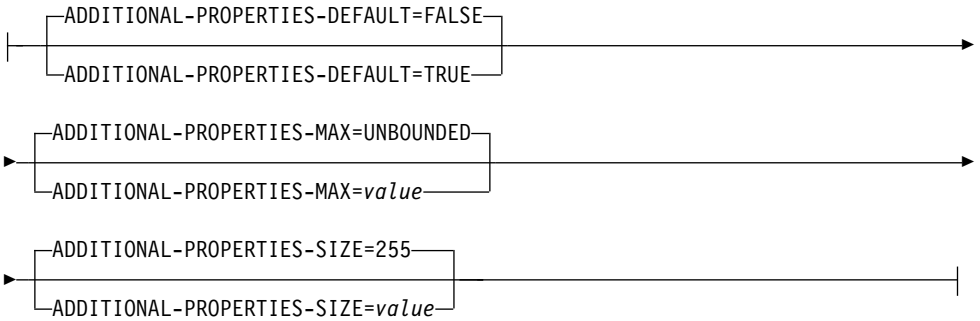
拡張データ・マッピング (マッピング・レベル **2.1** 以上):



拡張データ・マッピング (マッピング・レベル **3.0** 以上):



拡張データ・マッピング (マッピング・レベル **4.2** 以上):



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 (および使用する場合は、その継続文字) は、72 桁を超えて拡張することはできません。73 桁から 80 桁の間はブランクにする必要があります。

- 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。 アスタリスクの前のすべての部分 (スペースを含む) は、パラメーターの一部とみなされます。
- 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。
- 行の末尾の文字の位置にコンマがある場合、これはオプションの行分離文字であり、無視されます。

パラメーターの記述

ADDITIONAL-PROPERTIES-DEFAULT = { **true** | **false** }
追加プロパティーのサポートを明示的に宣言していない JSON スキーマ・オブジェクトが、それらのプロパティーをサポートしていると解釈されるかどうかを示します。追加の JSON プロパティーは、JSON スキーマ内で事前に定義されていない JSON オブジェクト内のプロパティーです。これらのプロパティーは、通常、予期しない追加データとしてデータ変換メカニズムによって拒否されます。**ADDITIONAL-PROPERTIES-DEFAULT** が TRUE に設定されている場合、または JSON スキーマによってオブジェクトに対して additionalProperties:true が明示的に設定されている場合、生成されるコピーブックにこれらの値を保持するためのスペースが割り当てられます。アプリケーションは、コピーブック内の関連するフィールドを使用して、それらの値と対話できます。

このパラメーターは、マッピング・レベル 4.2 以上で使用できます。

ADDITIONAL-PROPERTIES-MAX = { **0-20** | **UNBOUNDED** }
追加プロパティーをサポートする JSON オブジェクトに対してサポートされるこれらのプロパティーの数を示します。**ADDITIONAL-PROPERTIES-DEFAULT** を参照してください。生成されるコピーブックには、追加プロパティーのアドレッシングに適した構造が含まれます。デフォルトでは、サポートされるプロパティーの数に最大制約はありません。コピーブックは、制約のない配列と同様の方法で生成され、コンテナーを使用します。このパラメーターを **INLINE-MAXOCCURS-LIMIT** パラメーターと組み合わせて使用可能な最大制約を適用し、最大数のプロパティー用に固定長の配列を割り当てることができます。これにより、コンテナーは不用になります。

このパラメーターは、マッピング・レベル 4.2 以上で使用できます。

ADDITIONAL-PROPERTIES-SIZE = { **16-32767** | **255** }
各 JSON 追加プロパティーの最大サイズを示します。JSON オブジェクトが **ADDITIONAL-PROPERTIES-DEFAULT** によって定義された追加プロパティーをサポートする場合、生成されるコピーブックには、**ADDITIONAL-PROPERTIES-MAX** で指定された数までプロパティーをサポートするバインディングが設定されます。デフォルトでは、各追加プロパティーでサポートされる最大値は 255 文字です。そのサイズのフィールドは、生成されるコピーブックに生成されます。このサイズは、**ADDITIONAL-PROPERTIES-SIZE** パラメーターを設定することでカスタマイズできます。例えば、JSON オブジェクトは、以下のプロパティーを含むように処理されます。

"example": { "notes": "this extra property was not defined in the JSON Schema" }

追加プロパティーをサポートするようにコピーブックが生成されている場合は、その値全体がアプリケーションに渡されて処理されます。この値は、プロパティーのキーの前の先頭引用符で始まり、プロパティーの値の末尾の右中括弧で終わります。この例では約 100 文字です。**ADDITIONAL-PROPERTIES-SIZE** に使用する値は、考えられる最大の値を保持できる大きさにする必要があります。処理される値に対して割り当てられたバッファーが小さすぎると、エラー応答が生成されます。

このパラメーターは、マッピング・レベル 4.2 以上で使用できます。

BUNDLE = *value*

z/OS UNIX 上のバンドル・ディレクトリーのパスと名前を指定します。この値を指定する場合、JSON 支援機能によって、JSON バインディングがバンドル・ディレクトリー内に生成され、バンドル・マニフェストが作成されます。このパラメーターのパス情報は、**JSONTRANSFRM** パラメーターのパス情報を指定変更します。

CCSID = *value*

アプリケーション・データ構造に文字データをエンコードするために、実行時に使用される CCSID を指定します。このパラメーターの値は、**LOCALCCSID** システム初期設定パラメーターの値を指定変更します。 *value* は、Java および z/OS 変換サービスによってサポートされている EBCDIC CCSID である必要があります。このパラメーターを指定しない場合、アプリケーション・データ構造は、システム初期設定パラメーターで指定された CCSID を使用してエンコードされます。

このパラメーターは任意のマッピング・レベルで使用することができます。

CHAR-MULTIPLIER = { **1** | *value* }

マッピング・レベルが 1.2 以降のとき、各文字に許可されるバイト数を指定します。このパラメーターの *value* は、1 から 2,147,483,647 の範囲の正整数です。非数字に基づくマッピングはすべて、この乗数の対象です。バイナリー、数値、ゾーンおよびパック 10 進数フィールドは、この乗数の対象ではありません。

例えば、DBCS 文字を使用していて、実行時にすべての 2 バイト文字を潜在的なシフトアウトおよびシフトイン文字で囲むためのスペースを使用できるよう 3 の乗数を選択するときは、このパラメーターが便利です。

(UTF-16 を示す) **CCSID=1200** を設定する場合、**CHAR-MULTIPLIER** の有効値は 2 または 4 のみです。 UTF-16 を使用する場合のデフォルト値は 2 です。1 つの UTF-16 エンコード・ユニットを必要とする文字がアプリケーション・データに含まれると予期される場合は、**CHAR-MULTIPLIER=2** を使用してください。 2 つの UTF-16 エンコード・ユニットを必要とする文字がアプリケーション・データに含まれると予期される場合は、**CHAR-MULTIPLIER=4** を使用してください。

注: **CHAR-MULTIPLIER** を 1 に設定した場合、DBCS 文字は使用不可にはならず、2 に設定した場合、UTF-16 代理ペアは使用不可にはなりません。ただし、ワイド文字が定期的に使用される場合、いくつかの有効値は、割り振られたフィールドに入らなくなります。より大きな **CHAR-MULTIPLIER** 値を使用すると、XML で有効な文字数よりも多くの文字を、割り振られたフィールドに格納できます。該当する範囲制限を守るように注意する必要があります。

CHAR-VARYING = { **NO** | **NULL** | **YES** }

マッピング・レベルが 1.2 以上のとき、可変長文字データがマップされる方法を指定します。 可変長バイナリー・データ・タイプは、常にコンテナーまたは可変構造のいずれかにマップされます。 このパラメーターを指定しない場合、デフォルトのマッピングは、指定された言語によって決まります。 以下のオプションを選択できます。

NO 可変長文字データは固定長ストリングとしてマップされます。

NULL
可変長文字データはヌル終了ストリングにマップされます。

YES PL/I の場合、可変長文字データは CHAR VARYING データ型にマップされます。COBOL、C、および C++ 言語の場合、可変長文字データは、 2 つの関連エレメント (データ長およびデータ) で構成される同等の表現にマップされます。

CHAR-VARYING-LIMIT = { **32767** | *value* }

マッピング・レベルが 1.2 以上のとき、言語構造にマップされるバイナリー・データおよび可変長文字データの最大サイズを指定します。 文字データまたはバイナリー・データが、このパラメーターで指定される値より大きい場合は、コンテナーにマップされ、コンテナー名が生成された言語構造で使用されます。 *value* は 0 からデフォルトの 32 767 バイトの範囲で指定できます。

CHAR-WHITESPACE = **COLLAPSE** | **REPLACE** | **PRESERVE**

ストリング型の値に含まれる空白を CICS でどのように処理するかを指定します。

COLLAPSE
先行空白、後続空白、および埋め込み空白は除去され、タブ、改行、および連続スペースはすべて単一のスペース文字に置き換えられます。

REPLACE
タブまたは改行が適切な数のスペースに置換されます。

PRESERVE
データ値に含まれる空白がすべて保持されます。

CHAR-WHITESPACE パラメーターを設定しないと、空白が省略されます。

注: このパラメーターは、空白が常に省略される date-time、uri、base64Binary、または hexBinary 形式のフィールドには適用されません。

DATA-SCREENING = { **ENABLED** | **DISABLED** }

アプリケーション提供のデータのエラーを検査するかどうかを指定します。

ENABLED
言語構造と矛盾するアプリケーション提供のランタイム・データは、エラーとして扱われ、メッセージ DFHPI1010 が発行されます。エラー応答がアプリケーションに 返されます。

DISABLED
言語構造と矛盾するアプリケーション提供のランタイム・データの値は、 デフォルト値で置き換えられます。例えば、数値フィールド内のゼロは誤った値を置き換えます。メッセージ DFHPI1010 は発行され

ず、通常の応答がアプリケーションに返されます。この機能を使用して、 初期設定されていない出力フィールドから生成される INVALID_PACKED_DEC および INVALID_ZONED_DEC エラー応答を回避できます。

DATA-TRUNCATION = { **DISABLED** | **ENABLED** }
固定長フィールド構造で可変長データを許容するかどうかを指定します。

DISABLED
データが CICS が予期する固定長より短い場合に、CICS は切り捨てられたデータを拒否してエラー・メッセージを発行します。

ENABLED
データが CICS が予期する固定長より短い場合に、CICS は切り捨てられたデータを許容して、欠落データをヌル値として処理します。

DATETIME = { **PACKED15** | **STRING** }
JSON 日時プロパティーが CICS ABSTIME データ・フォーマットまたはテキストにマップされることを指定します。

PACKED15
JSON 日付プロパティー・フィールドは、CICS ABSTIME フォーマットにマップされます。

STRING
JSON 日時プロパティーはテキストにマップされます。このマッピングはこれまでのすべてのマッピング・レベルと同じです。

このパラメーターはマッピング・レベル 3.0 で使用できます。

DEFAULT-ARRAY-MAXITEMS = *value*
JSON スキーマで最大出現回数の情報 (maxItems) が示されていない場合に適用する最大配列境界を指定します。このパラメーターが設定されていない場合、最大制限は適用されません。このパラメーターの値は、1 から 2147483647 までの範囲の正整数です。このパラメーターを **INLINE-MAXOCCURS-LIMIT** パラメーターとともに使用することによって、 JSON 配列が言語構造にマップされる方法を制御することができます。

DEFAULT-CHAR-MAXLENGTH = { **255** | *value* }
マッピング・レベルが 1.2 以降のとき、JSON スキーマ文書に長さが暗黙指定されていない場合に、マッピングについて文字データのデフォルトの配列長を文字数で指定します。このパラメーターの *value* は、1 から 2 147 483 647 の範囲の正整数で指定できます。

DEFAULT-FRACTION-DIGITS = { **3** | *value* }
JSON 10 進スキーマ・タイプで使用するデフォルトの小数桁数を指定します。デフォルト値は 3 です。 COBOL の場合、有効な範囲は 0 から 17、またはパラメーター **WIDE-COMP3** が使用されている場合、0 から 30 です。 C または PLI の場合、有効な範囲は 0 から 30 までです。

INLINE-MAXOCCURS-LIMIT = { **1** | *value* }
インラインの可変反復内容を JSON スキーマ・キーワードの maxItems 属性に基づいて使用するかどうかを指定します。

INLINE-MAXOCCURS-LIMIT パラメーターは、マッピング・レベル 2.1 以降でのみ使用できます。 **INLINE-MAXOCCURS-LIMIT** の *value* は、0 から 32 767 の範囲

の正整数で指定できます。 値が 0 の場合は、インライン・マッピングが使用されないことを示します。 値が 1 の場合は、オプションのエレメントがインラインでマップされることを示します。 maxItems 属性の *value* が **INLINE-MAXOCCURS-LIMIT** の *value* より大きい場合、コンテナーに基づくマッピングが使用されます。それ以外の場合は、インライン・マッピングが使用されます。

可変反復リストをインラインでマップする場合は、繰り返されるデータの 1 つの項目の長さを考慮してください。長いインスタンスがほとんどない場合は、コンテナー・ベースのマッピングの方が望ましい方法です。短いインスタンスが多数ある場合は、インライン・マッピングの方が望ましい可能性があります。

JSONTRANSFRM = *value*
このパラメーターは、LINKable モードでは必須ですが、要求/応答モードおよび RESTful モードでは無効です。これは、CICS の **JSONTRANSFRM** バンドル・リソースに使用される名前を指定します。

JSON-SCHEMA = *value*
JSON スキーマの読み込み元になるファイルの、完全修飾された z/OS UNIX 名。 この名前が存在しない場合、DFHJS2LS はファイルを作成しますが、ディレクトリー構造は作成しません。

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 = PLI-OTHER
高水準言語構造のプログラム言語が Enterprise PL/I 以外の PL/I の水準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。

LOGFILE = *value*
DFHJS2LS 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 z/OS UNIX 名です。この名前が存在しない場合、DFHJS2LS はファイルを作成しますが、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しませんが、DFHJS2LS 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。

MAPPING-LEVEL = { 1.0 | 1.1 | 1.2 | 2.0 | 2.1 | 2.2 | 3.0 | 4.0 | 4.1 | 4.2 | 4.3 }
JSON バインディングおよび言語構造の生成時に支援機能で使用するマッピング・レベルを指定します。 使用可能な最新のマッピング・レベルを使用する必要があります。 DFHJS2LS の場合、マッピング・レベル 3.0 以上を使用する必要があります。

3.0 これは、DFHJS2LS で使用できる最小マッピング・レベルです。

4.0 UTF-16 を使用する場合には、CICS TS 5.2 以降の領域でこのマッピング・レベルを使用します。

4.1 切り捨て可能な配列をサポートするには、CICS TS 5.2 以降の領域でこのマッピング・レベルを使用します。

4.2 追加プロパティーについては、CICS TS 5.4 以降の領域でこのマッピング・レベルを使用します。

4.3 多次元配列をサポートするには、CICS TS 5.4 以降の領域でこのマッピング・レベルを使用します。

MAPPING-OVERRIDES = { SAME-AS-MAPPING-LEVEL |HYPHENS-AS-UNDERSCORES | INTEGER-AS-PIC9 | LESS-DUP-NAMES | UNDERSCORES-AS-HYPHENS | NO-ARRAY-NAME-INDEXING }

言語構造を生成するときの指定されたマッピング・レベルについて、デフォルトの動作を指定変更するかどうかを指定します。

SAME-AS-MAPPING-LEVEL

このパラメーターは、マッピング・レベルと同じスタイルで言語構造を生成します。これはデフォルトです。

HYPHENS-AS-UNDERSCORES

PL/I のみ。このパラメーターは、生成される PL/I 言語構造を読みやすくするために、JSON スキーマ内のすべてのハイフンを文字 X ではなく下線に変換します。詳しくは、JSON スキーマから PL/I へのマッピングを参照してください。このオプションは、マッピング・レベル 4.2 で自動的に有効になります。

INTEGER-AS-PIC9

COBOL および DFHJS2LS のみ。このパラメーターは、JSON スキーマからの整数値を、英数字ではなく数字として含む言語構造を生成します。このオプションは、マッピング・レベル 4.0 で自動的に有効になります。

LESS-DUP-NAMES

このパラメーターは、フィールドを直接参照できるように名前の末尾に _value を付けた非構造性の構造フィールド名を生成します。例えば、以下の PL/I 言語構造で、MAPPING-OVERRIDES=LESS-DUP-NAMES が指定されるとき、レベル 12 フィールドの streetName には接尾部に _value が付いています。

```
09 streetName,
12 streetName CHAR(255) VARYING
UNALIGNED,
12 filler BIT (7),
12 attr_nil_streetName_value BIT (1),
```

結果構造は以下のとおりです。

```
09 streetName,
12 streetName_value CHAR(255) VARYING
UNALIGNED,
12 filler BIT (7),
12 attr_nil_streetName_value BIT (1),
```

このオプションは、マッピング・レベル 4.2 で自動的に有効になります。

UNDERSCORES-AS-HYPHENS

COBOL のみ。 このパラメーターは、生成される COBOL 言語構造を読みやすくするために、JSON スキーマ内のすべての下線を文字 X ではなくハイフンに変換します。フィールド名の競合が発生する場合、そのフィールドが必ず固有になるように番号が付きます。 詳しくは、JSON スキーマから COBOL へのマッピングを参照してください。

このオプションは、マッピング・レベル 4.0 で自動的に有効になります。

NO-ARRAY-NAME-INDEXING

COBOL および Enterprise PL/I のみ。配列内のフィールド名が上位構造のスコープ内でのみ固有であるようにします。

MINIMUM-RUNTIME-LEVEL = { MINIMUM | 3.0 | 4.0 | 4.1 | 4.2 | 4.3 | CURRENT }

JSON バインディングをデプロイすることが可能な、CICS の最小ランタイム環境を指定します。 指定した他のパラメーターと一致しないレベルを選択すると、エラー・メッセージが送信されます。 選択可能なオプションは以下のとおりです。

MINIMUM

選択したパラメーターを前提として、最低限可能な CICS 実行時レベルが自動的に割り振られます。

3.0

CICS JSON 支援機能を使用して、高機能のデータ・マッピングを利用する場合は、ランタイム・レベル 3.0 以上を指定します。

4.0

生成された Web サービス・バインディング・ファイルは、CICS TS 5.2 以降の領域に正常にデプロイされます。この実行時レベルでは、**MAPPING-LEVEL** パラメーターにマッピング・レベル 4.0 以前を使用できます。このレベルでは任意のオプション・パラメーターを使用できます。

4.1

生成された Web サービス・バインディング・ファイルは、CICS TS 5.2 以降の領域に正常にデプロイされます。この実行時レベルでは、**MAPPING-LEVEL** パラメーターには、マッピング・レベル 4.1 以前を使用することができます。

4.2

生成された Web サービス・バインディング・ファイルは、CICS TS V5.4 以降の領域に正常にデプロイされます。この実行時レベルでは、**MAPPING-LEVEL** パラメーターには、マッピング・レベル 4.2 以前を使用することができます。

4.3

生成された Web サービス・バインディング・ファイルは、CICS TS 5.4 以降の領域に正常にデプロイされます。この実行時レベルでは、**MAPPING-LEVEL** パラメーターには、マッピング・レベル 4.3 以前を使用することができます。

CURRENT

生成されたバインディング・ファイルを、ランタイム環境がバインディング・ファイルの生成に使用された領域と同じである CICS 領域にデプロイする場合、このランタイム・レベルを使用します。

OVERWRITE-OUTPUT = { NO | YES }
ファイル・システム上の既存の CICS BUNDLE が上書き可能かどうかを制御します。

NO 既存 BUNDLE は何も置き換えられません。 既存 BUNDLE が検出された場合、DFHJS2LS はエラー・メッセージ DFHPI9689E を発行し強制終了します。

YES どの既存 BUNDLE も置き換えられます。 既存 BUNDLE が検出された場合、メッセージ DFHPI9683W が発行され、ファイルが置き換えられたことを通知します。

PDSCP = *value*
区分データ・セット・メンバーで使用されるコード・ページを指定します。ここで、*value* は CCSID 番号または Java コード・ページ番号です。 このパラメーターを指定しない場合、z/OS UNIX システム・サービス・コード・ページが使用されます。例えば、PDSCP=037 を指定することができます。

PDSLIB = *value*
生成された高水準言語が含まれる区分データ・セットの名前を指定します。

PDSMEM = *value*
高水準言語構造が含まれる区分データ・セット・メンバーの名前を生成するために DFHJS2LS が使用する、1 から 6 文字の文字接頭部を指定します。

DFHJS2LS 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。

STRUCTURE = { *PDSMEM_value* | *data* }
C および C++ での、最上位データ構造の名前。デフォルト値は **PDSMEM** パラメーターの値です。

WIDE-COMP3 = { **FULL** | **NO** | **YES** }
生成される COBOL または PL/I 言語構造でパック 10 進数の可変長の最大サイズを制御します。

FULL COBOL および PL/I の場合。DFHJS2LS は、すべての有効な値を保持できるサイズでパック 10 進数フィールドを生成します。最大サイズは 31 桁です。これはデフォルトです。

NO COBOL のみ。 DFHJS2LS は、COBOL 言語構造タイプ COMP-3 を生成しているとき、パック 10 進数の可変長を 18 に制限します。 パック 10 進数のサイズが 18 より大きい場合、指定されたタイプは合計 18 桁に制限されていることを示すメッセージ DFHPI9022W が出されます。

YES COBOL のみ。 DFHJS2LS は、COBOL 言語構造タイプ COMP-3 を生成しているとき、最大サイズ 31 をサポートします。

注: NO および YES オプションを指定すると、すべての有効な値を表わすことができないフィールドが生成されます。FULL オプションは、この問題を回避します。ただし、FULL オプションを指定すると、パック 10 進数フィールドで無効な値が表される可能性があります。例えば、スキーマに最大サイズとして最大 5 桁の数字と最大 2 桁の小数点以下の数値が指定されている場合、FULL オプシ

ョンを指定すると、7 桁の数字が許容されるパック 10 進数フィールドが生成されます。この場合、25000 や 999.99 などの有効な値を収容するだけのスペースがあると同時に、9999.99 などの無効な値を収容するだけのスペースも提供されることになります。FULL オプションを使用する場合は、アプリケーション・データに無効な値が生成されないよう注意してください。

例

```
//JS2LS JOB '
accounting information
',
name,MSGCLASS=A
// SET QT=''''
//JAVAPROG EXEC DFHJS2LS,
// TMPFILE=&QT.&SYSUID.&QT
/INPUT.SYSUT1 DD *
PDSLIB=//CICSHLQ.SDFHSAMP
PDSMEM=CPYBK2
JSON-SCHEMA=example.json
LANG=COBOL
LOGFILE=/u/exampleapp/example.log
MAPPING-LEVEL=4.0
CHAR-VARYING=NULL
JSONTRANSFRM=EXAMPLE
BUNDLE=/u/exampleapp/bundles/exampleBundle
/*
```

CICS JSON アシスタントのマッピング・レベル

マッピングとは、言語構造と JSON スキーマ間の情報の変換方法を指定する一連の規則です。 現在有効な最も高度なマッピングを活用するために、CICS アシスタントの **MAPPING-LEVEL** パラメーターを最新レベルに設定することをお勧めします。

マッピングの各レベルは、前のマッピングの機能を継承します。マッピングのレベルが一番高いと、一番優れた機能を利用できます。 マッピングの最高レベルでは、実行時のデータ変換がより強力に制御され、特定のデータ型や JSON プロパティーへのサポートの制限が取り外されています。

以前のレベルで有効だったアプリケーションを再展開したい場合には、**MAPPING-LEVEL** パラメーターをそのレベルに設定できます。

全マッピング・レベルでの制限

- JSON スキーマで使用されるデータ型は、明示的に宣言される必要があります。
- $ref を使用した、外部文書への JSON オブジェクト参照は、JSON スキーマ内ではサポートされません。

マッピング・レベル 4.3

マッピング・レベル 4.3 は、CICS TS V5.4 以上と互換性があります。

マッピング・レベル 4.3 は、主に DFHJS2LS で使用されますが、CICS Web サービス支援機能、XML 支援機能、および JSON 支援機能にも含まれています。このマッピング・レベルは、JSON での多次元配列のサポートを実装します。

マッピング・レベル 4.2

マッピング・レベル 4.2 は、CICS TS V5.4 以上と互換性があります。

マッピング・レベル 4.2 は、主に DFHJS2LS で使用されますが、CICS Web サービス支援機能、XML 支援機能、および JSON 支援機能にも含まれています。このマッピング・レベルは、JSON の追加プロパティーのサポートを実装し、DFHJS2LS に 3 つのパラメーター (**ADDITIONAL-PROPERTIES-DEFAULT**、**ADDITIONAL-PROPERTIES-MAX**、および **ADDITIONAL-PROPERTIES-SIZE**) を導入します。

マッピング・レベル 4.1

マッピング・レベル 4.1 は、CICS TS 5.3 以上の領域と互換性があります。

マッピング・レベル 4.1 は、CICS Web サービス支援機能、XML 支援機能、および JSON 支援機能に追加されています。このマッピング・レベルは、既存のコピーブックからボトムアップで生成される単純配列に対して改善されたマッピングを実装します。また、配列内で初期化されていない末尾のストレージの自動検出機能、およびそのようなレコードを生成される XML/JSON フォームから省略する機能を CICS に追加します。

DFHLS2WS、DFHWS2LS、DFHLS2SC、DFHSC2LS、DFHJS2LS、および DFHLS2JS は **TRUNCATE-NULL-ARRAYS** および **TRUNCATE-NULL-ARRAY-VALUES** パラメーターをサポートします。

TRUNCATE-NULL-ARRAY-VALUES に値を指定する場合、 **TRUNCATE-NULL-ARRAYS=ENABLED** も指定する必要があります。

マッピング・レベル 4.0

このマッピング・レベルでは、以下のサポートが提供されます。

マッピング・レベル 4.0 以上では、DFHLS2JS は、COBOL OCCURS DEPENDING ON 節をサポートし、COBOL 文字配列の JSON スキーマへのマッピングをサポートします。この動作は、CICS JSON 支援機能で **CHAR-OCCURS** パラメーターを使用することによって設定できます。

- パラメーター **DATA-TRUNCATION=ENABLED** を指定する必要があります。
- 複合 OCCURS DEPENDING ON はサポートされません。この制約は、構造体の最後のフィールドに対してのみ OCCURS DEPENDING ON がサポートされることを意味します。
- CICS は、OCCURS DEPENDING ON 節のターゲットとして修飾名 ('OF' キーワードの使用) をサポートしません (例えば、`FIELD1 OF STRUCTURE1`)。
- CICS は UNBOUNDED キーワードをサポートしません。アプリケーションによって予期される表の最大サイズにバインドされた整数を指定する必要があります。

マッピング・レベル 4.0 以上では、JSON Web サービスは、UTF-16 Unicode を使用してエンコードされるアプリケーション・データの変換をサポートします。

- LS2JS を使用する場合、UTF-16 に言語固有のデータ型を使用することによって、この動作を使用可能にできます。
- JS2LS を使用する場合、CCSID=1200 を設定することによって、この動作を使用可能にできます。

- CICS がサポートしている Unicode コード・ページは、「UTF-16BE with IBM Private Use Area」(CCSID 1200) のみです。
- UTF-8 を使用してエンコードされたアプリケーション・データの変換はサポートされていません。

注: DFHLS2JS は、COBOL GROUP USAGE NATIONAL 節をサポートしていません。

マッピング・レベル 3.0 以上

このマッピング・レベルでは、以下のサポートが提供されます。

- DFHJS2LS は、date-time データ型を CICS ASKTIME 形式にマップします。
- DFHLS2JS は、JSON スキーマおよび Web サービス・バインディングを、1 つのコンテナーのみでなく、多数のコンテナーを使用するアプリケーションから生成できます。
- 固定長データ構造で記述される切り捨てられたデータの許容。この動作は、CICS アシスタントで **DATA-TRUNCATION** パラメーターを使用することによって設定できます。

マッピング・レベル 2.2 以上

このオプションは、SOAP Web サービスとの互換性のために保持されています。JSON で使用することは推奨されていません。

マッピング・レベル 2.1 以上の場合

このオプションは、SOAP Web サービスとの互換性のために保持されています。JSON で使用することは推奨されていません。

このマッピング・レベルでは、新規の **INLINE-MAXOCCURS-LIMIT** パラメーターによって、および **CHAR-VARYING** パラメーターの新規の値によって可変コンテンツを処理する方法をさらに制御できます。

マッピング・レベル 2.1 以上では、DFHJS2LS は、配列に関する以下の新規サポート、および強化されたサポートを提供します。

- **INLINE-MAXOCCURS-LIMIT** パラメーター
- minItems プロパティー

INLINE-MAXOCCURS-LIMIT パラメーターは、可変の繰り返しリストがインラインにマップされるかどうかを指定します。 インラインでの可変反復内容のマップの詳細については、 324 ページの『DFHJS2LS 内のエレメントの可変配列』を参照してください。

マッピング・レベル 2.1 以上の場合、DFHLS2JS は以下の JSON マッピングをサポートします。

- COBOL および PL/I での FILLER フィールドは無視されます
- **CHAR-VARYING** パラメーターに対する COLLAPSE 値
- **CHAR-VARYING** パラメーターに対する BINARY 値

COBOL および PL/I の FILLER フィールドは無視され、生成された JSON スキーマには表示されず、実行時に、データ構造に適切なギャップが残ります。

COLLAPSE は、 CICS がテキスト・フィールドで末尾スペースを無視するように します。

BINARY は 2 進数フィールドのサポートを提供します。この値は、COBOL を JSON スキーマに変換する際に役立ちます。 このオプションは SBCS 文字配列のみで使用可能です。このオプションにより、配列を通常のストリングではなく、Base64 エンコード・データを含む固定長 JSON ストリングにマップできます。

マッピング・レベル 1.2 以上の場合

このオプションは、SOAP Web サービスとの互換性のために保持されています。JSON で使用することは推奨されていません。

以下に示すバッチ・ツールの追加パラメーターを使用して、実行時の文字データおよびバイナリー・データの変換方法をさらに制御できます。

- **CHAR-VARYING**
- **CHAR-VARYING-LIMIT**
- **CHAR-MULTIPLIER**
- **DEFAULT-CHAR-MAXLENGTH**

DFHJS2LS で **CHAR-MULTIPLIER** パラメーターを使用する場合は、このパラメーターの値が文字データに必要なスペース量の計算に使用された後、以下の規則が適用されることに注意してください。

- DFHJS2LS はこれらのマッピングを提供します。
  - 最大長が 32 767 バイトより大きい可変長の文字データ・タイプはコンテナーにマップされます。 下限は **CHAR-VARYING-LIMIT** パラメーターを使用して設定できます。 コンテナーの名前を格納するために、16 バイトのフィールドが言語構造に作成されます。 実行時に、文字データはコンテナーに格納され、コンテナー名は言語構造に入ります。
  - 最大長が 32 768 バイトより小さい可変長の文字データ・タイプは、C/C++ および Enterprise PL/I を除くすべての言語で VARYING 構造にマップされます。C/C++ ではこれらのデータはヌル終了ストリングにマップされ、Enterprise PL/I ではこれらのデータ・タイプは VARYINGZ 構造にマップされます。 **CHAR-VARYING** パラメーターを使用して、可変長の文字データがマップされる方法を選択できます。
  - 最大長が 32 768 バイトより小さい可変長のバイナリー・データは、すべての言語で VARYING 構造にマップされます。 最大長が 32 768 バイト以上である場合、データはコンテナーにマップされます。 コンテナーの名前を格納するために、16 バイトのフィールドが言語構造に作成されます。 実行時に、バイナリー・データはコンテナーに格納され、コンテナー名は言語構造に入ります。

これらに関連付けられた長さを持たない文字データ型が JSON スキーマに存在する場合は、DFHJS2LS の **DEFAULT-CHAR-MAXLENGTH** パラメーターを使用して、デフォルトの長さを割り当てることができます。

DFHLS2JS はこれらのマッピングを提供します。

- 文字フィールドは string データ型にマップされ、実行時に固定長フィールドまたはヌル終了ストリングとして処理できます。PL/I を除くすべての言語での、実行時の可変長文字データの処理方法を、**CHAR-VARYING** パラメーターを使用して選択できます。
- Base64Binary データ・タイプはデータの最大長が 32 767 バイトより大きいか、長さが定義されていない場合に、コンテナーにマップされます。 データの長さが 32 767 以下である場合は、base64Binary データ・タイプがすべての言語で VARYING 構造にマップされます。

マッピング・レベル 1.1 以上の場合

このオプションは、SOAP Web サービスとの互換性のために保持されています。JSON で使用することは推奨されていません。

このマッピング・レベルでは、JSON 文字とバイナリー・データ型のマッピングが強化されています(特に、JSON スキーマで異なる値で定義された maxLength プロパティーと minLength プロパティーを持つ可変長のデータをマップする場合)。 データは、以下のように処理されます。

- 16 MB より大きい固定長を持つ文字およびバイナリー・データ・タイプが、PL/I を除くすべての言語でコンテナーにマップされます。PL/I では、32 767 バイトより大きい固定長の文字およびバイナリー・データ・タイプがコンテナーにマップされます。 コンテナーの名前を格納するために、16 バイトのフィールドが言語構造に作成されます。 実行時に、固定長データはコンテナーに格納され、コンテナー名は言語構造に入ります。

  コンテナーの長さは可変的なので、コンテナーにマップされた固定長データと、JSON スキーマで指定された固定長が一致するように、スペースやヌルが埋め込まれたり切り捨てられたりすることはありません。 データ長が重要な場合、アプリケーションに書き込んでデータ長を確認するか、または CICS 領域で妥当性検査をオンにすることができます。妥当性検査は、パフォーマンスに大きな影響を及ぼします。

マッピング・レベル 1.1 のみ

このオプションは、SOAP Web サービスとの互換性のために保持されています。JSON で使用することは推奨されていません。

このマッピング・レベルでは、JSON 文字とバイナリー・データ型のマッピングが強化されています(特に、JSON スキーマで異なる値で定義された maxLength プロパティーと minLength プロパティーを持つ可変長のデータをマップする場合)。 データは次のように処理されます。

- 可変長のバイナリー・データ・タイプがコンテナーにマップされます。 コンテナーの名前を格納するために、16 バイトのフィールドが言語構造に作成されます。 実行時に、バイナリー・データはコンテナーに格納され、コンテナー名は言語構造に入ります。
- 最大長が 32 767 バイトより大きい可変長の文字データ・タイプはコンテナーにマップされます。 コンテナーの名前を格納するために、16 バイトのフィールドが言語構造に作成されます。 実行時に、文字データはコンテナーに格納され、コンテナー名は言語構造に入ります。

- 固定長が 16 MB 未満の文字とバイナリー・データ型は、PL/I 以外のすべての言語の固定長フィールドにマップされます。 PL/I の場合、32 767 バイト以下の固定長の文字とバイナリー・データ型は固定長フィールドにマップされます。
- CICS エンコード・データおよびデコード・データは base64Binary 形式ではなく hexBinary 形式で処理されます。 JSON スキーマの Base64Binary データ型は、言語構造内のフィールドにマップされます。 フィールドのサイズは次の公式を使用して計算されます。4×(ceil(*z*/3))。各部の意味は次のとおりです。
  - *z* は JSON スキーマ内のデータ型の長さです。
  - ceil(*x*) は *x* 以上の最小の整数です。

  z の長さが 24 566 バイトより大きい場合、結果として生成される言語構造のコンパイルは失敗します。 24 566 バイトより大きい base64Binary データがある場合は、マッピング・レベル 1.2 の使用をお勧めします。 マッピング・レベル 1.2 では、言語構造内のフィールドを使用する代わりに、base64Binary データをコンテナーにマップできます。

マッピング・レベル 1.0 のみ

このオプションは、SOAP Web サービスとの互換性のために保持されています。JSON で使用することは推奨されていません。

次の制限事項 (以後のマッピング・レベルでは変更されている) に注意してください。

- DFHJS2LS は、JSON スキーマ内の文字およびバイナリー・データ型を、言語構造内の固定長フィールドにマップします。 この部分的な JSON スキーマを見てください。

  ```
  "example":{
  "type":"string",
  "maxLength":33000
  ```

  この部分的な JSON スキーマは、COBOL 言語構造で次のように表示されます。

  ```
  15 example PIC X(33000)
  ```
- CICS エンコード・データおよびデコード・データは base64Binary 形式ではなく hexBinary 形式で処理されます。 DFHJS2LS は、Base64Binary データを固定長文字フィールドにマップします。フィールドの内容は、アプリケーション・プログラムによってエンコードまたはデコードされる必要があります。
- DFHLS2JS は、言語構造内の文字フィールドとバイナリー・フィールドを固定長フィールドとして解釈し、これらのフィールドを、maxLength プロパティーを持つ JSON ストリングにマップします。十分なデータがない場合には、実行時に言語構造内のフィールドがスペースやヌルで埋められます。

高水準言語と JSON のスキーマ・マッピング

CICS JSON アシスタントを使用して、高水準言語構造と JSON スキーマとの間のマッピングを生成します。また、CICS JSON アシスタントは、高水準言語データ構造から JSON スキーマを生成したり、JSON スキーマから高水準言語データ構造を生成したりします。

ユーティリティー・プログラム DFHJS2LS と DFHLS2JS を合わせて、CICS JSON アシスタントといいます。

- DFHLS2JS は、高水準言語構造を JSON スキーマにマッピングします。
- DFHJS2LS は、JSON スキーマを高水準言語構造にマッピングします。

以下に示すように、2 つのマッピングは対称的ではありません。

- DFHLS2JS を使用して言語データ構造を処理し、結果として生成される JSON スキーマを DFHJS2LS で処理する場合、最終的なデータ構造と元の構造が同じであると期待することはできません。
- DFHJS2LS を使用して JSON スキーマを処理し、結果として生成された言語構造を DFHLS2JS を使用して処理する場合、最終的な JSON スキーマと元のスキーマが同じであると期待することはできません。
- 場合によっては、DFHJS2LS により、DFHLS2JS でサポートされていない言語構造が生成されます。

DFHLS2JS によって処理される高水準言語構造は、CICS によってサポートされ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コーディングする必要があります。

COBOL から JSON スキーマへのマッピング:

DFHLS2JS ユーティリティー・プログラムは、COBOL データ構造と JSON スキーマ定義との間のマッピングをサポートしています。

COBOL の名前は、以下の規則に従って JSON の名前に変換されます。

- 重複する名前は、1 つ以上の数字が追加されて固有の名前になります。

  例えば、`year` の 2 つのインスタンスは `year` と `year1` になります。
- ハイフンは、下線に置き換えられます。一連の連続するハイフンは、連続する下線で置き換えられます。

  例えば、`current-user--id` は `current_user__id` になります。
- ハイフンで区切られており、大文字のみを含む名前のセグメントは、小文字に変換されます。

  例えば、`CA-REQUEST-ID` は `ca_request_id` になります。
- 数字で始まる名前には、先頭に下線が追加されます。

  例えば、`9A-REQUEST-ID` は `_9a_request_id` になります。

CICS 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 データ記述エレメントは、DFHLS2JS 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 および 77 のデータ記述項目はサポートされていません。 レベル番号が 88 のデータ記述項目は無視されます。
- データ記述項目における以下の節は、サポートされません。

  REDEFINES

  RENAMES (レベル 66)

DATE FORMAT

- データ記述項目における以下の節は、無視されます。

  BLANK WHEN ZERO

  JUSTIFIED

  VALUE

- SIGN 文節 SIGN TRAILING はサポートされます。 SIGN 文節 SIGN LEADING は、DFHLS2JS で指定されたマッピング・レベルが 1.2 以上の場合のみサポートされます。
- SIGN TRAILING 文節と SIGN LEADING 文節の両方では、SEPARATE CHARACTER はマッピング・レベル 1.2 以上でサポートされます。
- USAGE 節における以下の句は、サポートされません。

  OBJECT REFERENCE

  POINTER

  FUNCTION-POINTER

  PROCEDURE-POINTER

- USAGE 文節の次の句は、マッピング・レベル 1.2 以上でサポートされます。

  COMPUTATIONAL-1

  COMPUTATIONAL-2

- 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、S、および Z です。
- PACKED-DECIMAL データ記述項目でサポートされる PICTURE 文字は、9、S、V、および Z です。
- 編集済みの数値データ記述項目でサポートされている PICTURE 文字は、9 と Z だけです。
- MAPPING-LEVEL パラメーターを 1.2 以上に設定して、CHAR-VARYING パラメーターを NULL に設定すると、文字配列は `string` にマップされ、ヌル終了ストリングとして処理されます。
- MAPPING-LEVEL パラメーターを 1.2 以上に設定して、CHAR-VARYING パラメーターを BINARY に設定すると、文字配列は `string` にマップされ、バイナリー・データとして処理されます。
- MAPPING-LEVEL パラメーターを 1.2 以上に設定して、CHAR-VARYING パラメーターを COLLAPSE に設定すると、ストリングの末尾の空白文字は無視されます。
- OCCURS DEPENDING ON 節は、マッピング・レベル 4.0 以上でサポートされます。複雑な OCCURS DEPENDING ON はサポートされません。つまり、構造の最後のフィールドにのみ OCCURS DEPENDING ON がサポートされます。
- OCCURS INDEXED BY 節は、すべてのマッピング・レベルでサポートされます。

COBOL のデータ記述	**JSON** スキーマ定義
PIC X(*n*) PIC A(*n*) PIC G(*n*) DISPLAY-1 PIC N(*n*)	"type":"string", "maxLength": *n*
PIC S9 DISPLAY PIC S99 DISPLAY PIC S999 DISPLAY PIC S9999 DISPLAY PIC S9(*n*) DISPLAY PIC S9(*n*) COMP PIC S9(*n*) COMP-4 PIC S9(*n*) COMP-5 PIC S9(*n*) BINARY	"type":"integer", "minimum":- (*n* + 1), "maximum": *n* 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 DISPLAY PIC 99 DISPLAY PIC 999 DISPLAY PIC 9999 DISPLAY PIC 9(*n*) DISPLAY PIC 9(*n*) COMP PIC 9(*n*) COMP-4 PIC 9(*n*) COMP-5 PIC 9(*n*) BINARY	"type":"integer", "minimum":0, "maximum": *n* 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	**JSON** スキーマ定義
PIC S9(*m*)V9(*n*) COMP-3	"type":"number", "format":"decimal", "minimum": *x* , "maximum": *y* , "multipleOf": *z* 各部の意味は次のとおりです。 *x* は、文字「9」のパターンで表現できる最小値です。 *y* は、文字「9」のパターンで表現できる最大値です。 *z* は、使用可能な最小単位 = 1 / 10^n です。
PIC 9(*m*)V9(*n*) COMP-3	"type":"number", "format":"decimal", "minimum": 0, "maximum": *y* , "multipleOf": *z* 各部の意味は次のとおりです。 *y* は、文字「9」のパターンで表現できる最大値です。 *z* は、使用可能な最小単位 = 1 / 10^n です。
PIC S9(*15*) COMP-3 DATETIME=PACKED15 の場合マッピング・レベル 3.0 でサポートされる	"type":"string", "format":"date-time" タイム・スタンプのフォーマットは、RFC3339 により定義されます。
PIC S9(*m*)V9(*n*) DISPLAY マッピング・レベル 1.2 以上でサポートされる	"type":"number", "format":"decimal", "minimum": *x* , "maximum": *y* , "multipleOf": *z* 各部の意味は次のとおりです。 *x* は、文字「9」のパターンで表現できる最小値です。 *y* は、文字「9」のパターンで表現できる最大値です。 *z* は、使用可能な最小単位 = 1 / 10^n です。
COMP-1 マッピング・レベル 1.2 以上でサポートされる **注:** IBM Hexadecimal Floating Point (HFP) データ表記は、JSON に使用される IEEE-754-1985 表記とまったく同じではありません。一部の値が 1 つ表記からもう 1 つの表記に正確に変換されない可能性があります。 極端に大きいまたは小さい値は、float データ型に対して有効ではない場合があります。 HFP 表記との間の変換時に、一部の値で精度が失われる可能性があります。正確な変換が重要になる場合、COMP-1 データ型を使用せずに、固定精度のデータ型を代わりに使用することを検討してください。	"type":"number", "format":"float"
COMP-2 マッピング・レベル 1.2 以上でサポートされる **注:** IBM Hexadecimal Floating Point (HFP) データ表記は、JSON に使用される IEEE-754-1985 表記とまったく同じではありません。一部の値が 1 つ表記からもう 1 つの表記に正確に変換されない可能性があります。 極端に大きいまたは小さい値は、double データ型に対して有効ではない場合があります。 HFP 表記との間の変換時に、一部の値で精度が失われる可能性があります。正確な変換が重要になる場合、COMP-2 データ型を使用せずに、固定精度のデータ型を代わりに使用することを検討してください。	"type":"number", "format":"double"

COBOL のデータ記述	**JSON** スキーマ定義

data description OCCURS *n* TIMES	マッピング・レベル **4.0** 以下
	プリミティブの場合:
	"type":"array"
	"maxItems":
	n
	"minItems":
	n
	"items":{
	"type":"object",
	"properties":{
	name
	:{
	data description JSON
	}
	}
	"required":[
	name
	]
	}
	構造の場合:
	"type":"array"
	"maxItems":
	n
	"minItems":
	n
	"items":{
	data description JSON
	}
	ここで *data description JSON* は、COBOL *data description* の JSON スキーマ表記です。 *name* は、COBOL *data description* の名前です。
	マッピング・レベル **4.1** 以上
	TRUNCATE-NULL-ARRAYS = *DISABLED*
	構造化配列およびプリミティブ配列はどちらも以下のようにマップされます。
	"type":"array"
	"maxItems":
	n
	"minItems":
	n
	"items":{
	data description JSON
	}
	TRUNCATE-NULL-ARRAYS = *ENABLED*
	プリミティブ配列は上記のようにマップされ、構造化配列は以下のようにマップされます。
	"type":"array"
	"maxItems":
	n
	"minItems":
	0
	"items":{
	data description JSON
	}

data description
OCCURS
n
TO
m
TIMES
DEPENDING ON
t
マッピング・レベル 4.0 でサポート
	"field-name"
	:{
	"type":
	"array"
	,
	"maxItems":
	m
	"minItems":
	n
	"items":{
	...
	}
	}
	配列項目の内容は、使用されるデータ型によって異なります。

PIC X OCCURS
n
TIMES
PIC A OCCURS
n
TIMES
PIC G DISPLAY-1 OCCURS
n
TIMES
PIC N OCCURS
n
TIMES
	CHAR-OCCURS =STRING の場合:
	"field-name"
	:{
	"type":
	"string"
	,
	"maxLength":
	n
	}
	これはストリングです。

COBOL のデータ記述	**JSON** スキーマ定義

	CHAR-OCCURS =ARRAY の場合:
	
	マッピング・レベル **4.0** 以下
	
	"field-name"
	:{
	"maxItems":
	m
	,
	"minItems":
	n
	,
	"items":{
	"type":
	"object"
	,
	"properties":{
	"field-name":{
	"type":
	"string"
	,
	"maxLength":1
	}
	},
	"required":[
	"field-name"
	]
	}
	}
	
	これは単一文字の配列です。
	
	マッピング・レベル **4.1** 以上
	
	TRUNCATE-NULL-ARRAYS = *DISABLED*
	
	構造化配列およびプリミティブ配列はどちらも以下のようにマップされます。
	
	"type":"array"
	"maxItems":
	n
	"minItems":
	n
	"items":{
	data description JSON
	}
	
	TRUNCATE-NULL-ARRAYS = *ENABLED*
	
	プリミティブ配列は上記のようにマップされ、構造化配列は以下のようにマップされます。
	
	"type":"array"
	"maxItems":
	n
	"minItems":
	0
	"items":{
	data description JSON
	}

PIC X OCCURS
n
TO
m
TIMES
DEPENDING ON
 	
t
PIC A OCCURS
n
TO
m
TIMES
DEPENDING ON
 	
t
PIC G DISPLAY-1 OCCURS
n
TO
m
TIMES
DEPENDING ON
 	
t
PIC N OCCURS
n
TO
m
TIMES
DEPENDING ON
 	
t
	CHAR-OCCURS =STRING の場合:
	
	"field-name"
	:{
	"type":
	"string"
	,
	"maxLength":
	m
	"minLength":
	n
	}

COBOL のデータ記述	JSON スキーマ定義
PIC N(*n*) USAGE NATIONAL **CHAR-USAGE** =NATIONAL の場合: PIC N(*n*)	マッピング・レベル 4.0 以上の場合: "type":"string", "maxLength": *n* 実行時に、CICS はアプリケーション・データ構造フィールドに UTF-16 データを取り込みます。

JSON スキーマから **COBOL** へのマッピング**:**

DFHJS2LS ユーティリティー・プログラムは、JSON スキーマと COBOL データ構造との間のマッピングをサポートしています。

CICS 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 変数の固有かつ有効なフィールド名を生成します。

1. COBOL 予約語には接頭部「X」が付きます。

   例えば、 DISPLAY は XDISPLAY となります。

2. A から Z、a から z、0 から 9、またはハイフン以外の文字は、「X」で置き換えられます。

   例えば、monthly_total は monthlyXtotal になります。

3. 最後の文字がハイフンである場合は、「X」で置き換えられます。

   例えば、ca-request は ca-requestX となります。

4. 同じスコープ内の重複した名前は、名前の 2 番目以降のインスタンスに 1 つまたは 2 つの数字を追加することによって固有にします。

   例えば、 year の 3 つのインスタンスは year 、 year1 、および year2 になります。

   もし上記の動作が望ましくないなら、ユーザーはユーティリティーの入力として MAPPING-OVERRIDES=NO-ARRAY-NAME-INDEXING を指定できます。これにより、名前の 2 番目以降のインスタンスへの 1 つまたは 2 つの数字の追加が無効になります。

5. JSON スキーマは、"type" 値が "array" で、キーワード "minItems" と "maxItems" が省略されるか、それぞれ別の値が指定されている場合には、変数の基数が可変になるように指定します。変数の基数が可変になるようスキーマで指定されている場合、フィールド名は接尾部 "_cont" および "_num" が付けられて作成されます。

   詳しくは、 324 ページの『DFHJS2LS 内のエレメントの可変配列』を参照してください。

6. JSON スキーマは、"required" キーワード配列 (周りを囲む JSON スキーマ "object" タイプに関連付けられている) に指定されていない変数がオプションとなるように指定します。オプション・フィールドの場合、追加のフィールドは、エレメント名に接尾部 _num が追加されて生成されます。実行時にこれは、JSON データに値が存在しなかったことを示す場合はゼロ、JSON データに値が存在した場合は非ゼロになります。

7. フィールド名は 28 文字に限定されます。接頭部および接尾部を含めて生成された名前がこの長さを超えると、エレメント名が切り捨てられます。

DFHJS2LS は、指定されたマッピング・レベルを使用して、スキーマ・タイプを次の表に従って、 COBOL のデータ記述エレメントにマップします。次の点に注意してください。

- **MAPPING-LEVEL** パラメーターを 1.2 以上に設定して、**CHAR-VARYING** パラメーターを NULL に設定すると、可変長文字データはヌル終了ストリングにマップされ、ヌル終止符として追加された 1 つの文字が割り振られます。
- **MAPPING-LEVEL** パラメーターを 1.2 以上に設定して、**CHAR-VARYING** パラメーターを YES に設定すると、可変長文字データは 2 つの関連エレメント (長さフィールドとデータ・フィールド) にマップされます。以下に例を示します。

```
"textString": {
"type":"string",
"maxLength":10000,
"minLength":1
}
```

これは、次にマップします。

```
15 textString-length PIC S9999 COMP-5 SYNC
15 textString PIC X(10000)
```

JSON スキーマ・キーワード	**COBOL** のデータ記述

以下のすべて:
 	
`"type":"null"`
`"type": []`
	サポートされていない

`"$schema":"http://json-schema.org/draft-04/schema#"`	このキーワードは無視されますが、ドラフト 04 JSON スキーマ仕様と互換性があることが想定されています。
`"title": "same text"` `"description": "more text"`	これらのキーワードは無視されます。
`"format": "<predefined values>"`	"format" キーワードは、生成された構造またはランタイム値のいずれかを変更するために使用されます。サポートされる "format" の使用方法については、この表の後半にある情報を参照してください。

`"type":`
`"array",`
`"items": {<JSON Sub-schema>},`
`"additionalItems": false,`
`"maxItems":`
m
`,`
`"minItems":`
n
	多次元配列は、マッピング・レベル 4.3 以上でサポートされ、混合タイプ配列はサポートされません。
	
	"additionalItems" は、false になることが想定されています。他にサポートされている値はありません。
	
	"minItems" と "maxItems" の両方が存在し、それらが等しい場合、配列は固定基数として処理されます。それ以外の場合、配列は可変基数として処理されます。 324 ページの『DFHJS2LS 内のエレメントの可変配列』を参照してください。

`"type": "array",` `"uniqueItems": true`	"uniqueItems" は、JSON 配列でサポートされていません。

`"type": "object",`
`"additionalProperties": false,`
`"properties": { [ "<element name>": {<JSON Sub-schema>} [,] ]* }`
`"required": [ [ "<element name>" [,] ]* ]`
	現在サポートされている JSON オブジェクトの唯一の形式は、名前付きエレメントの固定セットです。
	
	これは、エレメント名を使用する構造 (または副構造) を生成します。
	
	"additionalProperties" は、値が **ADDITIONAL-PROPERTIES-DEFAULT** パラメーターで設定されていない場合、false と想定されます。
	
	"required" 配列にないか、"required" 配列が存在しない場合、"properties" オブジェクト内のエレメントはすべて "optional" と見なされます。 "optional" エレメントにはゼロから X までの可変オーディナリティーが提供されます (X は 1 または配列の項目の最大数、その項目は配列として定義)。 324 ページの『DFHJS2LS 内のエレメントの可変配列』を参照してください。

`"type": "object",` `"additionalProperties": true`	オブジェクトは前の例のようにマップされ、追加プロパティーをサポートする追加フィールドがあります。"additionalProperties" プロパティーは、**ADDITIONAL-PROPERTIES-DEFAULT** パラメーターで設定されていない場合、false と想定されます。 有効な場合、**ADDITIONAL-PROPERTIES-MAX** パラメーターで指定された値までスペースが割り当てられます。各スペースの文字数は、**ADDITIONAL-PROPERTIES-SIZE** パラメーターで設定されています。個々のプロパティーは、PIC X(*z*) フィールドにマップされます。z は、**ADDITIONAL-PROPERTIES-SIZE** パラメーターによって定義されます。**ADDITIONAL-PROPERTIES-MAX** の値が 0 より大きい場合、プロパティーは、**maxItems** が設定された配列のプロパティーと同じようにマップされます。 **注:** CICS での z/OS Connect の使用など、CICS での JSON サポートを構成する方法はいくつかあります。古い CICS Java パイプライン・テクノロジーを使用する場合、**追加プロパティー**は、com.ibm.cics.json.enableAxis2Handlers JVM システム・プロパティーが true に設定されていないときにのみサポートされます。

`"type": "object",`
`"maxProperties": m,`
`"minProperties": n,`
`"patternProperties": {},`
`"dependencies":`
	これらのキーワードは、いずれも JSON オブジェクトではサポートされません。

JSON スキーマ・キーワード	**COBOL** のデータ記述
"type":"string" "maxLength": *m* "pattern": "regular expression>", "minLength": *l*	PIC X(*z*) ここで *z* の値は *m* に基づいていますが、 *CHAR-VARYING* パラメーターの設定に依存します。 *m* は "maxLength" キーワードに基づき、固定長ストリングとして扱われます。 "pattern" および "minLength" の制約事項は、コメントとしてのみ言語構造に渡されます。
"type":"string" "maxLength": *m*	マッピング・レベル 4.0 以上で CCSID=1200 の場合: PIC N(*z*) USAGE NATIONAL ここで *z* の値は *m* に基づいていますが、 **CHAR-VARYING** パラメーターの設定に依存します。 *m* は "maxLength" キーワードに基づき、固定長ストリングとして扱われます。

"*name*":{
"type":"string",
"format":"date-time"
}
	PIC S9(15) COMP-3
	
	DATETIME=PACKED15 の場合、すべてサポートされます。
	
	"maxLength" および "minLength" は、この形式ではサポートされないことに注意してください。

"*name*":{
"type":"string",
"format":"uri"
}
	PIC X(*m*)
	
	ここで、*m* は "maxLength" キーワードに基づき、固定長ストリングとして扱われます。
	
	マッピング・レベル 4.0 以上で **CCSID=1200** の場合:
	
	PIC N(m) USAGE NATIONAL
	
	ここで、*m* は "maxLength" キーワードに基づき、固定長ストリングとして扱われます。

"*name*":{
"type":"string",
"format":"base64Binary"
}
	PIC X(*m*)
	
	ここで、 *m* は "maxLength" キーワードに基づきます。

"*name*":{
"type":"string",
"format":"hexBinary"
}
	PIC X(*m*)
	
	ここで、 *m* は "maxLength" キーワードに基づきます。

"*name*":{
"type":"string",
"format":"<predefined>"
}
	PIC X(*m*)
	
	ここで、 *m* は "maxLength" キーワードに基づき固定長ストリングとして扱われます。 また、 <predefined> は *email*、 *hostname*、 *ipv4*、 または *ipv6* のいずれかです。関連する "pattern" が使用され、コメントに渡されます。
	
	マッピング・レベル 4.0 以上で **CCSID=1200** の場合:
	
	PIC N(*m*) USAGE NATIONAL
	
	ここで、 *m* は "maxLength" キーワードに基づき固定長ストリングとして扱われます。 また、 <predefined> は *email*、 *hostname*、 *ipv4*、 または *ipv6* のいずれかです。関連する "pattern" が使用され、コメントに渡されます。

"type":"boolean"	PIC X DISPLAY 値 x'00' は false、x'01' は true を意味します。
"type": "integer", "exclusiveMaximum": true, "exclusiveMinimum": true, "multipleOf": n	"exclusiveMaximum" および "exclusiveMinimum" の制約事項は、コメントとしてのみ言語構造に渡されます。 "multipleOf" は無視されます。
"type"="integer", minimum=0, maximum=255	マッピング・レベル 3.0 以下: PIC X DISPLAY マッピング・レベル 4.0 以上 (または、マッピング・レベルとは関係なく、INTEGER-AS-PIC9 パラメーターが指定されている場合): PIC 9(*z*) COMP-5 SYNC または PIC 9(*z*) DISPLAY ここで $10^{(z-1)} < m <= 10^{z}$

JSON スキーマ・キーワード	**COBOL** のデータ記述
"type":"integer", minimum:-128, maximum:127	マッピング・レベル 3.0 以下: PIC X DISPLAY マッピング・レベル 4.0 以上 (または、マッピング・レベルとは関係なく、INTEGER-AS-PIC9 パラメーターが指定されている場合): PIC S9(*z*) COMP-5 SYNC または PIC S9(*z*) DISPLAY ここで $10^{(z-1)} < m <= 10^{z}$
"type":"integer", minimum:0, maximum; *m*	PIC 9(*z*) COMP-5 SYNC または PIC 9(*z*) DISPLAY ここで $10^{(z-1)} < m <= 10^{z}$
"type":"integer", minimum:- *m* , maximum: *m* -1	PIC S9(*z*) COMP-5 SYNC または PIC S9(*z*) DISPLAY ここで $10^{(z-1)} < m <= 10^{z}$
"type": "number", "maximum": m, "minimum": n, "exclusiveMaximum": true, "exclusiveMinimum": true, "multipleOf": n	"maximum"、"minimum"、"exclusiveMaximum" および "exclusiveMinimum" の制約事項は、コメントとしてのみ言語構造に渡されます。 "multipleOf" は無視されます。
"type":"number" "format":"decimal"	PIC 9(*p*)V9(*n*) COMP-3 ここで、*p* および *n*はデフォルト値です。
"type":"number" "format":"float"	マッピング・レベル 1.1 以下: • PIC X(32) マッピング・レベル 1.2 以上の場合 • COMP-1 **注:** IBM Hexadecimal Floating Point (HFP) データ表記は、JSON に使用される IEEE-754-1985 表記とまったく同じではありません。一部の値が 1 つ表記からもう 1 つの表記に正確に変換されない可能性があります。 極端に大きいまたは小さい値は、float データ型に対して有効ではない場合があります。 HFP 表記との間の変換時に、一部の値で精度が失われる可能性があります。正確な変換が重要になる場合、COMP-1 データ型を使用せずに、固定精度のデータ型を代わりに使用することを検討してください。
"type":"number" "format":"double"	マッピング・レベル 1.1 以下: • PIC X(32) マッピング・レベル 1.2 以上の場合 • COMP-2 **注:** IBM Hexadecimal Floating Point (HFP) データ表記は、JSON に使用される IEEE-754-1985 表記とまったく同じではありません。一部の値が 1 つ表記からもう 1 つの表記に正確に変換されない可能性があります。 極端に大きいまたは小さい値は、double データ型に対して有効ではない場合があります。 HFP 表記との間の変換時に、一部の値で精度が失われる可能性があります。正確な変換が重要になる場合、COMP-2 データ型を使用せずに、固定精度のデータ型を代わりに使用することを検討してください。

JSON スキーマ・キーワード	COBOL のデータ記述

`oneOf: [ { "properties": { "A": {...} } } ,`
`{ "properties": { "B": {...} } } ]`
`anyOf: [ { "properties": { "A": {...} } } ,`
`{ "properties": { "B": {...} } } ]`
`allOf: [ { "properties": { "A": {...} } } ,`
`{ "properties": { "B": {...} } } ]`
	オプションの配列を使用した論理パスは、{ "properties": { "A":..., "B":... } } というタイプの単一オブジェクトが定義されている場合と同様にマージされます。JSON プロパティーはそれぞれオプションとして扱われます。同じプロパティー名の競合する定義がサブオプションに含まれている場合は、エラー・メッセージが発行されます。例えば、単一プロパティーがストリング (パス A 内) とオブジェクト (パス B 内) の両方として定義されている場合、そのような定義はサポートされず、エラー・メッセージが表示されます。
	
	JSON スキーマで複雑な論理構造を定義できますが、複雑な論理構造内で暗黙的に指定された細かい点は、言語構造へのマッピングで失われる可能性があります。変換プロセスは、スキーマからの組み合わせルールを強制しようとしません。指定された JSON プロパティーが存在するか存在しないかを示す言語構造フィールドとのみ対話します。
	
	サブオプションに同じプロパティー名で互換性のある定義が含まれている場合、DFHJS2LS は、制約の関連パターンをマージしようとしますが、プロセスではいくつかの細かい点が失われる場合があります。例えば、以下の定義を想定します。
	
	`"A": { "oneOf": [ { "type": "string", "maxLength": 5}, {"type": "string", "minLength": 7, "maxLength": 8} ] }`
	
	『A』 は、最大 5 文字の長さのストリング、または長さが 7 から 8 文字のストリングのいずれかとして定義されています。構成のマージにより、6 文字の長さが無効であることは認識されず、0 から 8 文字のストリングにマッピングされる可能性があります。
	
	論理構成を含むほとんどのシナリオでは単純な言語構造にマップされますが、複雑な論理構成では、マッピング処理中に妥協が図られることがあります。最良の結果を得るために、JSON スキーマで論理構成を使用せずに、同じ JSON プロパティーに代替宣言を定義してください。

注: CICS では、signed long の最大値 (2^63 - 1) より大きい整数値を変換できません。ただし、整数値が引用符で囲まれている場合は変換できます。

注: 数値型のスキーマに指定された最小値および最大値は、COBOL データ型にマップされる目的でのみ使用されます。実行時にデータがこれらの値に基づいて検証されることはありません。

表に示したスキーマ・タイプの一部は、`minimum` キーワードおよび `maximum` キーワードに指定された値 (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。

  ```
  "maximum":
  a
  "minimum":
  -a
  ```

  ここで、*a* は数字 9 からなるストリングです。

- 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。

  ```
  "maximum":
  a
  "minimum":0
  ```

  ここで、*a* は数字 9 からなるストリングです。

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++** から **JSON** スキーマへのマッピング:

DFHLS2JS ユーティリティー・プログラムは、C および C++ データ・タイプと JSON スキーマ定義との間のマッピングをサポートしています。

C および C++ の名前は、次の規則に従って JSON の名前に変換されます。

1. JSON プロパティー名に含まれる無効な文字は、「X」で置き換えられます。

   例えば、 monthly-total は monthlyXtotal となります。
2. 重複する名前は、1 つ以上の数字が追加されて固有の名前になります。

   例えば、year の 2 つのインスタンスは year と year1 になります。

DFHLS2JS 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++ のタイプは DFHLS2JS ではサポートされません。 _Packed 修飾子が構造用にサポートされています。 以下の制限が適用されます。

- ヘッダー・ファイルには、最上位の struct インスタンスが含まれていなければなりません。
- 自身をメンバーとして含む構造化タイプを宣言することはできません。
- 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
  long double
  wchar_t (C++ のみ)
- 以下は、ヘッダー・ファイル内に存在する場合、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)
  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、以下の項目を指定できません。
  共用体
  クラス宣言
  列挙型データ・タイプ
  ポインター型変数
  テンプレート宣言
  定義済みマクロ (名前の先頭と末尾が 2 つの下線文字 (__) 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、32 ビットのコンパイラーを使用して int が 4 バイトに確実にマップされるようにする必要があります。
- 次の C++ 予約済みキーワードはサポートされません。

  explicit

  using

  namespace

  typename

  typeid
- **MAPPING-LEVEL** パラメーターを 1.2 以上に設定して、**CHAR-VARYING** パラメーターを NULL に設定すると、文字配列は string にマップされ、ヌル終了ストリングとして処理されます。
- **MAPPING-LEVEL** パラメーターを 1.2 以上に設定して、**CHAR-VARYING** パラメーターを BINARY に設定すると、文字配列は xsd:base64Binary にマップされ、バイナリー・データとして処理されます。
- **MAPPING-LEVEL** パラメーターを 1.2 以上に設定して、**CHAR-VARYING** パラメーターを COLLAPSE に設定すると、<xsd:whiteSpace value="collapse"/> がストリング用に生成されます。

C および **C++** のデータ・タイプ	スキーマの **simpleType**
char[*z*]	"type":"string" "maxlength": *z*
char16_t[*n*]	マッピング・レベル 4.0 以上の場合: "type":"string" "maxlength": *n* 実行時に、CICS はアプリケーション・データ構造フィールドに UTF-16 データを取り込みます。
char[8] DATETIME=PACKED15 の場合マッピング・レベル 3.0 以上でサポートされる	"type":"string" "format":"date-time" タイム・スタンプのフォーマットは、RFC3339 により定義されます。
char short int long long long	"type":"integer", "minimum":- (*n* + 1), "maximum": *n* ここで、*n* はプリミティブで表すことができる最大値です。
unsigned char unsigned short unsigned int unsigned long unsigned long long	"type":"integer", "minimum":0, "maximum": *n* ここで、*n* はプリミティブで表すことができる最大値です。

C および **C++** のデータ・タイプ	スキーマの **simpleType**
`bool` (C++ のみ)	`"type":"boolean"`

`float`
 	
マッピング・レベル 1.2 以上でサポートされる
	`"type":"number",`
	`"format":"float"`
	
	注: IBM Hexadecimal Floating Point (HFP) データ表記は、JSON に使用される IEEE-754-1985 表記とまったく同じではありません。一部の値が、ある表記から別の表記に正確に変換されない可能性があります。極端に大きいまたは小さい値は、float データ型に対して有効ではない場合があります。HFP 表記との間の変換時に、一部の値で精度が失われる可能性があります。正確な変換が重要になる場合、float データ型を使用せずに、固定精度のデータ型を代わりに使用することを検討してください。

`double`
 	
マッピング・レベル 1.2 以上でサポートされる
	`"type":"number",`
	`"format":"double"`
	
	注: IBM Hexadecimal Floating Point (HFP) データ表記は、JSON に使用される IEEE-754-1985 表記とまったく同じではありません。一部の値が、ある表記から別の表記に正確に変換されない可能性があります。極端に大きいまたは小さい値は、double データ型に対して有効ではない場合があります。HFP 表記との間の変換時に、一部の値で精度が失われる可能性があります。正確な変換が重要になる場合、double データ型を使用せずに、固定精度のデータ型を代わりに使用することを検討してください。

type name[*n*]	プリミティブの場合:
	
	`"type":"array"`
	`"maxItems":`
	`n`
	`"minItems":`
	`n`
	`"items":{`
	`"type":"object",`
	`"properties":{`
	`name`
	`:{`
	`type JSON`
	`}`
	`}`
	`"required":[`
	`name`
	`]`
	`}`
	
	構造体の場合:
	
	`"type":"array"`
	`"maxItems":`
	`n`
	`"minItems":`
	`n`
	`"items":{`
	`type JSON`
	`}`
	
	ここで *type JSON* は、C または C++ 型の JSON スキーマ表記です。

JSON スキーマから **C** および **C++** へのマッピング**:**

DFHJS2LS ユーティリティー・プログラムは、JSON スキーマと、C および C++ データ・タイプとの間のマッピングをサポートしています。

CICS 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 変数および C++ 変数の固有かつ有効なフィールド名を生成します。

1. A から Z、a から z、0 から 9、または _ 以外の文字は、「X」で置き換えられます。

   例えば、monthly-total は monthlyXtotal になります。
2. 先頭文字が英字ではない場合は、先頭文字が「X」で置き換えられます。

   例えば、_monthlysummary は Xmonthlysummary になります。
3. 同じスコープ内の重複した名前は、名前の 2 番目以降のインスタンスに 1 つまたは 2 つの数字を追加することによって固有にします。

   例えば、 year の 3 つのインスタンスは year 、 year1 、および year2 になります。
4. JSON スキーマは、"type" 値が "array" で、キーワード "minItems" と "maxItems" が省略されるか、それぞれ別の値が指定されている場合には、変数の基数が可変になるように指定します。変数の基数が可変になるようスキーマで指定されている場合、フィールド名は接尾部 "_cont" および "_num" が付けられて作成されます。

   詳しくは、 324 ページの『DFHJS2LS 内のエレメントの可変配列』を参照してください。
5. JSON スキーマは、"required" キーワード配列 (周りを囲む JSON スキーマ "object" タイプに関連付けられている) に指定されていない変数がオプションとなるように指定します。 オプション・フィールドの場合、追加のフィールドは、エレメント名に接尾部 _num が追加されて生成されます。実行時にこれは、JSON データに値が存在しなかったことを示す場合はゼロ、JSON データに値が存在した場合は非ゼロになります。
6. フィールド名は 50 文字までに制限されています。接頭部および接尾部を含めて生成された名前がこの長さを超えると、エレメント名が切り捨てられます。

DFHJS2LS は、次の表に従って JSON スキーマ・タイプ値を C および C++ データ・タイプにマップします。次の規則も適用されます。

- **MAPPING-LEVEL** パラメーターを 1.2 以上に設定して、**CHAR-VARYING** パラメーターを NULL に設定すると、可変長文字データはヌル終了ストリングにマップされ、ヌル終止符として追加された 1 つの文字が割り振られます。
- **MAPPING-LEVEL** パラメーターを 1.2 以上に設定して、**CHAR-VARYING** パラメーターを YES に設定すると、可変長文字データは 2 つの関連エレメント (長さフィールドとデータ・フィールド) にマップされます。

JSON スキーマ・キーワード	**C** および **C++** のデータ・タイプ

以下のすべて:
`"type":"null"`
`"type": []`
	サポートされていない

JSON スキーマ・キーワード	**C** および **C++** のデータ・タイプ
`"$schema":"http://json-schema.org/draft-04/schema#"`	このキーワードは無視されますが、ドラフト 04 JSON スキーマ仕様と互換性があることが想定されています。
`"title": "same text"` `"description": "more text"`	これらのキーワードは無視されます。
`"format": "<predefined values>"`	"format" キーワードは、生成された構造またはランタイム値のいずれかを変更するために使用されます。サポートされる "format" の使用方法については、以下を参照してください。

`"type":`
`"array",`
`"items": {<JSON Sub-schema>},`
`"additionalItems": false,`
`"maxItems":`
m
`,`
`"minItems":`
n
	多次元配列は、マッピング・レベル 4.3 以上でサポートされ、混合タイプ配列はサポートされません。
	
	"additionalItems" は、false になることが想定されています。他にサポートされている値はありません。
	
	"minItems" と "maxItems" の両方が存在し、それらが等しい場合、配列は固定基数として処理されます。それ以外の場合、配列は可変基数として処理されます。 324 ページの『DFHJS2LS 内のエレメントの可変配列』を参照してください。

`"type": "array",` `"uniqueItems": true`	"uniqueItems" は、JSON 配列でサポートされていません。 <JSON Sub-schema> はサポートされている "type" を定義する必要がありますが、"type" を "array" にすることはできません。 これは、生成される言語構造に対する制約事項です。

`"type": "object",`
`"additionalProperties": false,`
`"properties": { [ "<element name>": {<JSON Sub-schema>} [,] ]* }`
`"required": [ [ "<element name>" [,] ]* ]`
	現在サポートされている JSON オブジェクトの唯一の形式は、名前付きエレメントの固定セットです。
	
	これは、エレメント名を使用して構造 (または副構造) を生成します。
	
	"additionalProperties" は、値が **ADDITIONAL-PROPERTIES-DEFAULT** パラメーターで設定されていない場合、false と想定されます。
	
	"required" 配列にないか、"required" 配列が存在しない場合、"properties" オブジェクト内のエレメントはすべて "optional" と見なされます。 "optional" エレメントにはゼロから X までの可変オーディナリティーが提供されます (X は 1 または配列の項目の最大数、その項目は配列として定義)。 324 ページの『DFHJS2LS 内のエレメントの可変配列』を参照してください。

`"type": "object",` `"additionalProperties": true`	オブジェクトは前の例のようにマップされ、追加プロパティーをサポートする追加フィールドがあります。"additionalProperties" プロパティーは、**ADDITIONAL-PROPERTIES-DEFAULT** パラメーターで設定されていない場合、false と想定されます。 有効な場合、**ADDITIONAL-PROPERTIES-MAX** パラメーターで指定された値までスペースが割り当てられます。各スペースの文字数は、**ADDITIONAL-PROPERTIES-SIZE** パラメーターで設定されています。個々のプロパティーは、PIC X(*z*) フィールドにマップされます。*z* は、**ADDITIONAL-PROPERTIES-SIZE** パラメーターによって定義されます。**ADDITIONAL-PROPERTIES-MAX** の値が 0 より大きい場合、プロパティーは、**maxItems** が設定された配列のプロパティーと同じようにマップされます。**注:** CICS での z/OS Connect の使用など、CICS での JSON サポートを構成する方法はいくつかあります。古い CICS Java パイプライン・テクノロジーを使用する場合、**追加プロパティー**は、com.ibm.cics.json.enableAxis2Handlers JVM システム・プロパティーが true に設定されていないときにのみサポートされます。

`"type": "object",`
`"maxProperties": m,`
`"minProperties": n,`
`"patternProperties": {},`
`"dependencies":`
	これらのキーワードは、いずれも JSON オブジェクトではサポートされません。

`"type":"string"`
`"maxLength":`
m
`"pattern": "regular expression>",`
`"minLength":`
l
	char[*z*]
	
	ここで *z* の値は *m* に基づいていますが、 **CHAR-VARYING** パラメーターの設定に依存します。
	
	m は "maxLength" キーワードに基づき、固定長ストリングとして扱われます。
	
	"pattern" および "minLength" の制約事項は、コメントとしてのみ言語構造に渡されます。

`"type":"string"`
`"maxLength":`
m
	マッピング・レベル 4.0 以上で CCSID=1200 の場合:
	
	char16_t[*z*]
	
	ここで *z* の値は *m* に基づいていますが、 **CHAR-VARYING** パラメーターの設定に依存します。
	
	m は "maxLength" キーワードに基づき、固定長ストリングとして扱われます。

JSON スキーマ・キーワード	**C** および **C++** のデータ・タイプ

"*name*":{
"type":"string",
"format":"date-time"
}
	char[*8*]
	
	DATETIME=PACKED15 の場合、すべてサポートされます。

"*name*":{
"type":"string",
"format":"uri"
}
	char[*m*]
	
	ここで、*m* は "maxLength" キーワードに基づき、固定長ストリングとして扱われます。
	
	マッピング・レベル 4.0 以上で **CCSID=1200** の場合:
	
	char16_t[*m*]
	
	ここで、*m* は "maxLength" キーワードに基づき、固定長ストリングとして扱われます。

"*name*":{
"type":"string",
"format":"base64Binary"
}
	char[*m*]
	
	ここで、*m* は "maxLength" キーワードに基づいています。

"*name*":{
"type":"string",
"format":"hexBinary"
}
	char[*m*]
	
	ここで、*m* は "maxLength" キーワードに基づいています。

"*name*":{
"type":"string",
"format":"<predefined>"
}
	char[*m*]
	
	ここで、 *m* は "maxLength" キーワードに基づき固定長ストリングとして扱われます。 また、 <predefined> は *email*、 *hostname*、 *ipv4*、 または *ipv6* のいずれかです。関連する "pattern" が使用され、コメントに渡されます。
	
	マッピング・レベル 4.0 以上で **CCSID=1200** の場合:
	
	char16_t[*m*]
	
	ここで、 *m* は "maxLength" キーワードに基づき固定長ストリングとして扱われます。 また、 <predefined> は *email*、 *hostname*、 *ipv4*、 または *ipv6* のいずれかです。関連する "pattern" が使用され、コメントに渡されます。

"type":"boolean"	bool (C++ のみ) short (C のみ)
"type": "integer", "exclusiveMaximum": true, "exclusiveMinimum": true, "multipleOf": n	"exclusiveMaximum" および "exclusiveMinimum" の制約事項は、コメントとしてのみ言語構造に渡されます。 "multipleOf" は無視されます。
"type":"integer", minimum:-128, maximum:127	signed char
"type":"integer", minimum:0, maximum:255	unsigned char
"type":"integer", minimum:-32768, maximum:32767	short
"type":"integer", minimum:0, maximum:65535	unsigned short
"type":"integer", minimum:-2147483648, maximum:2147483647	int
"type":"integer", minimum:0, maximum:4294967295	unsigned int
"type":"integer", minimum:-9223372036854775808, maximum:9223372036854775807	long long
"type":"integer", minimum:0, maximum:18446744073709551615	unsigned long long
"type": "number", "maximum": m, "minimum": n, "exclusiveMaximum": true, "exclusiveMinimum": true, "multipleOf": n	"maximum"、"minimum"、"exclusiveMaximum" および "exclusiveMinimum" の制約事項は、コメントとしてのみ言語構造に渡されます。 "multipleOf" は無視されます。

JSON スキーマ・キーワード	**C** および **C++** のデータ・タイプ
`"type":"number"` `"format":"float"`	マッピング・レベル 1.1 以下: • `char[32]` マッピング・レベル 1.2 以上の場合 • `float(*)` **注:** IBM Hexadecimal Floating Point (HFP) データ表記は、JSON に使用される IEEE-754-1985 表記とまったく同じではありません。一部の値が、ある表記から別の表記に正確に変換されない可能性があります。極端に大きいまたは小さい値は、float データ型に対して有効ではない場合があります。 HFP 表記との間の変換時に、一部の値で精度が失われる可能性があります。正確な変換が重要になる場合、float データ型を使用せずに、固定精度のデータ型を代わりに使用することを検討してください。
`"type":"number"` `"format":"double"`	マッピング・レベル 1.0 以下: • `char[32]` マッピング・レベル 1.2 以上の場合 • `double(*)` **注:** IBM Hexadecimal Floating Point (HFP) データ表記は、JSON に使用される IEEE-754-1985 表記とまったく同じではありません。一部の値が、ある表記から別の表記に正確に変換されない可能性があります。極端に大きいまたは小さい値は、double データ型に対して有効ではない場合があります。 HFP 表記との間の変換時に、一部の値で精度が失われる可能性があります。正確な変換が重要になる場合、double データ型を使用せずに、固定精度のデータ型を代わりに使用することを検討してください。

`oneOf: [ { "properties": { "A": {...} } } ,`
`{ "properties": { "B": {...} } } ]`
`anyOf: [ { "properties": { "A": {...} } } ,`
`{ "properties": { "B": {...} } } ]`
`allOf: [ { "properties": { "A": {...} } } ,`
`{ "properties": { "B": {...} } } ]`
	オプションの配列を使用した論理パスは、`{ "properties": { "A":..., "B":... } }` というタイプの単一オブジェクトが定義されている場合と同様にマージされます。JSON プロパティーはそれぞれオプションとして扱われます。同じプロパティー名の競合する定義がサブオプションに含まれている場合は、エラー・メッセージが発行されます。例えば、単一プロパティーがストリング (パス A 内) とオブジェクト (パス B 内) の両方として定義されている場合、そのような定義はサポートされず、エラー・メッセージが表示されます。
	JSON スキーマで複雑な論理構造を定義できますが、複雑な論理構造内で暗黙的に指定された細かい点は、言語構造へのマッピングで失われる可能性があります。変換プロセスは、スキーマからの組み合わせルールを強制しようとしません。指定された JSON プロパティーが存在するか存在しないかを示す言語構造フィールドとのみ対話します。
	サブオプションに同じプロパティー名で互換性のある定義が含まれている場合、DFHJS2LS は、制約の関連パターンをマージしようとしますが、プロセスではいくつかの細かい点が失われる場合があります。例えば、以下の定義を想定します。
	`"A": { "oneOf": [ { "type": "string", "maxLength": 5}, {"type": "string", "minLength": 7, "maxLength": 8} ] }`
	『A』 は、最大 5 文字の長さのストリング、または長さが 7 から 8 文字のストリングのいずれかとして定義されています。構成のマージにより、6 文字の長さが無効であることは認識されず、0 から 8 文字のストリングにマッピングされる可能性があります。
	論理構成を含むほとんどのシナリオでは単純な言語構造にマップされますが、複雑な論理構成では、マッピング処理中に妥協が図られることがあります。最良の結果を得るために、JSON スキーマで論理構成を使用せずに、同じ JSON プロパティーに代替宣言を定義してください。

注**:** CICS では、signed long の最大値 (2^63 - 1) より大きい整数値を変換できません。ただし、整数値が引用符で囲まれている場合は変換できます。

注**:** 数値型のスキーマに指定された最小値および最大値は、C または C++ データ型にマップされる目的でのみ使用されます。実行時にデータがこれらの値に基づいて検証されることはありません。

PL/I から **JSON** スキーマへのマッピング**:**

DFHLS2JS ユーティリティー・プログラムは、PL/I データ構造と JSON スキーマ定義との間のマッピングをサポートしています。Enterprise PL/I コンパイラーと古

い PL/I コンパイラーの間には相違点があるため、PLI-ENTERPRISE と PLI-OTHER の 2 つの言語オプションがサポートされます。

PL/I の名前は、次の規則に従って JSON の名前に変換されます。

1. JSON プロパティー名に含まれる無効な文字は、「x」で置き換えられます。

   例えば、monthly$total は monthlyxtotal になります。

2. 重複する名前は、1 つ以上の数字が追加されて固有の名前になります。

   例えば、year の 2 つのインスタンスは year と year1 になります。

DFHLS2JS は次の表に従って PL/I 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 PL/I タイプは DFHLS2JS でサポートされません。 次の制約事項も適用されます。

- COMPLEX 属性を持つデータ項目はサポートされません。
- FLOAT 属性を持つデータ項目は、マッピング・レベル 1.2 以上でサポートされます。 Enterprise PL/I FLOAT IEEE はサポートされません。
- VARYING および VARYINGZ の純粋な DBCS ストリングは、マッピング・レベル 1.2 以上でサポートされます。
- 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。
- PICTURE ストリングはサポートされません。
- ORDINAL データ項目は、FIXED BINARY(7) データ・タイプとして扱われます。
- **MAPPING-LEVEL** パラメーターを 1.2 以上に設定して、**CHAR-VARYING** パラメーターを NULL に設定すると、文字配列は string にマップされ、ヌル終了ストリングとして処理されます。このマッピングは Enterprise PL/I には適用されません。
- **MAPPING-LEVEL** パラメーターを 1.2 以上に設定して、**CHAR-VARYING** パラメーターを BINARY に設定すると、文字配列は string にマップされ、バイナリー・データとして処理されます。
- **MAPPING-LEVEL** パラメーターが 1.2 以上に設定されていて、**CHAR-VARYING** パラメーターが COLLAPSE に設定されていると、先頭と末尾の空白文字が削除され、複数のスペースがシングル・スペースに置換されます。

DFHLS2JS は、PL/I の埋め込みアルゴリズムを完全には実装しないため、データ構造で埋め込みバイトを明示的に宣言する必要があります。 DFHLS2JS は、埋め込みバイトがないことを検出すると、メッセージを発行します。それぞれの最上位構造は、ダブルワード境界で開始して、構造内のそれぞれのバイトは正しい境界にマップされている必要があります。 次のコード・フラグメントについて考えてみます。

```
 3 FIELD1 FIXED BINARY(7),
3 FIELD2 FIXED BINARY(31),
3 FIELD3 FIXED BINARY(63);
```

この例では以下のようになります。
- FIELD1 の長さは 1 バイトで、任意の境界に合わせることができます。
- FIELD2 の長さは 4 バイトで、フルワード境界に合わせる必要があります。
- FIELD3 の長さは 8 バイトで、ダブルワード境界に合わせる必要があります。

Enterprise PL/I コンパイラーはフィールドを以下の順序に調整します。

1. FIELD3 の境界要件が最も厳しいため、FIELD3 が最初に配置されます。
2. FIELD2 は FIELD3 の直前のフルワード境界に合わせます。
3. FIELD1 が FIELD3 の直前のバイト境界に合わせます。

最後に、構造全体がフルワード境界に合うように、コンパイラーは FIELD1 の直前に 3 つの埋め込みバイトを挿入します。

DFHLS2JS は同等の埋め込みバイトを挿入しないため、DFHLS2JS が構造を処理する前にこれらの埋め込みバイトを明示的に宣言する必要があります。 以下に例を示します。

```
 3 PAD1 FIXED BINARY(7),
3 PAD2 FIXED BINARY(7),
3 PAD3 FIXED BINARY(7),
3 FIELD1 FIXED BINARY(7),
3 FIELD2 FIXED BINARY(31),
3 FIELD3 FIXED BINARY(63);
```

あるいは、すべてのフィールドを位置合わせされないと宣言するよう構造を変更して、この構造を使用するアプリケーションを再コンパイルすることもできます。 PL/I 構造上のメモリー位置合わせ要件について詳しくは、Enterprise PL/I for z/OS の製品情報を参照してください。

PL/I のデータ記述	**JSON** スキーマ定義
FIXED BINARY (*n*)	"type":"integer", "minimum":- (*n* + 1), "maximum": *n* ここで、*n* はプリミティブで表すことができる最大値です。
UNSIGNED FIXED BINARY(*n*) **制約事項:** Enterprise PL/I のみ	"type":"integer", "minimum":0, "maximum": *n* ここで、*n* はプリミティブで表すことができる最大値です。

PL/I のデータ記述	**JSON** スキーマ定義
FIXED DECIMAL(*n*,*m*)	"type":"number", "format":"decimal", "multipleOf": *x* , "maximum": *y* , "minimum":- *z* ここで、 *x* は、使用可能な最小単位 = 1 / 10^m です。 *y* は、*n* と *m* の組み合わせで表現できる最大値です。 *z* は、*n* と *m* の組み合わせで表現できる最大値です。
FIXED DECIMAL(15) **DATETIME=PACKED15** の場合マッピング・レベル 3.0 以上でサポートされる	"type":"string", "format":"date-time" タイム・スタンプのフォーマットは、RFC3339 により定義されます。
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。	"type":"string" "maxLength": *m* ここで、 *m* = *n* /8
CHARACTER(*n*) VARYING および VARYINGZ はマッピング・レベル 1.2 以上でもサポートされる **制約事項:** VARYINGZ は Enterprise PL/I でのみサポートされる	"type":"string" "maxLength": *n*
GRAPHIC(*n*) VARYING および VARYINGZ はマッピング・レベル 1.2 以上でもサポートされる **制約事項:** VARYINGZ は Enterprise PL/I でのみサポートされる	マッピング・レベル 1.0 および 1.1 で、 *m* = 2**n* の場合: "type":"string" "maxLength": *m* マッピング・レベル 1.2 以上の場合: "type":"string" "maxLength": *n*

PL/I のデータ記述	**JSON** スキーマ定義

WIDECHAR(*n*)
制約事項: Enterprise PL/I のみ
	マッピング・レベル 1.0 および 1.1 で、 *m* = 2*n* の場合:
	`"type":"string"`
	`"maxLength":`
	m
	
	マッピング・レベル 1.2 以上の場合:
	`"type":"string"`
	`"maxLength":`
	n
	
	マッピング・レベル 4.0 以上では、CICS はアプリケーション・データ構造フィールドに UTF-16 データを取り込みます。
	`<xsd:simpleType>`
	`<xsd:restriction base="xsd:string">`
	`<xsd:maxlength value="n"/>`
	`<xsd:whiteSpace value="preserve"/>`
	`</xsd:restriction>`
	`</xsd:simpleType>`

ORDINAL
制約事項: Enterprise PL/I のみ
	`"type":"integer",`
	`"minimum":0,`
	`"maximum":255`

BINARY FLOAT(*n*) ここで、*n* <= 21 マッピング・レベル 1.2 以上でサポートされる **注:** IBM Hexadecimal Floating Point (HFP) データ表記は、JSON に使用される IEEE-754-1985 表記とまったく同じではありません。一部の値が、ある表記から別の表記に正確に変換されない可能性があります。極端に大きいまたは小さい値は、float データ型に対して有効ではない場合があります。 HFP 表記との間の変換時に、一部の値で精度が失われる可能性があります。正確な変換が重要になる場合、BINARY FLOAT データ型を使用せずに、固定精度のデータ型を代わりに使用することを検討してください。	`"type":"number",` `"format":"float"`

PL/I のデータ記述	**JSON** スキーマ定義
`BINARY FLOAT( n )` ここで、`21 < n <= 53` 53 より大きい値はサポートされない。 マッピング・レベル 1.2 以上でサポートされる **注:** IBM Hexadecimal Floating Point (HFP) データ表記は、JSON に使用される IEEE-754-1985 表記とまったく同じではありません。一部の値が、ある表記から別の表記に正確に変換されない可能性があります。極端に大きいまたは小さい値は、float データ型に対して有効ではない場合があります。 HFP 表記との間の変換時に、一部の値で精度が失われる可能性があります。正確な変換が重要になる場合、BINARY FLOAT データ型を使用せずに、固定精度のデータ型を代わりに使用することを検討してください。	`"type":"number",` `"format":"double"`
`DECIMAL FLOAT( n )` ここで、`n <= 6` です。 マッピング・レベル 1.2 以上でサポートされる **注:** IBM Hexadecimal Floating Point (HFP) データ表記は、JSON に使用される IEEE-754-1985 表記とまったく同じではありません。一部の値が、ある表記から別の表記に正確に変換されない可能性があります。極端に大きいまたは小さい値は、float データ型に対して有効ではない場合があります。 HFP 表記との間の変換時に、一部の値で精度が失われる可能性があります。正確な変換が重要になる場合、DECIMAL FLOAT データ型を使用せずに、固定精度のデータ型を代わりに使用することを検討してください。	`"type":"number",` `"format":"float"`

PL/I のデータ記述	**JSON** スキーマ定義
DECIMAL FLOAT(*n*) ここで、6 < *n* <= 16 16 より大きい値はサポートされない。 マッピング・レベル 1.2 以上でサポートされる **注:** IBM Hexadecimal Floating Point (HFP) データ表記は、JSON に使用される IEEE-754-1985 表記とまったく同じではありません。一部の値が、ある表記から別の表記に正確に変換されない可能性があります。極端に大きいまたは小さい値は、float データ型に対して有効ではない場合があります。 HFP 表記との間の変換時に、一部の値で精度が失われる可能性があります。正確な変換が重要になる場合、DECIMAL FLOAT データ型を使用せずに、固定精度のデータ型を代わりに使用することを検討してください。	"type":"number", "format":"double"

name (*n*) *data description*	プリミティブの場合:
	
	"type":"array"
	"maxItems":
	n
	"minItems":
	n
	"items":{
	"type":"object",
	"properties":{
	name
	:{
	data description JSON
	}
	}
	"required":[
	name
	]
	}
	
	データ宣言の場合:
	
	"type":"array"
	"maxItems":
	n
	"minItems":
	n
	"items":{
	data description JSON
	}
	
	ここで、*data description JSON* は、PL/I データ記述の JSON スキーマ表記です。

JSON スキーマから **PL/I** へのマッピング**:**

DFHJS2LS ユーティリティー・プログラムは、JSON スキーマと PL/I データ構造との間のマッピングをサポートしています。Enterprise PL/I コンパイラーと古い PL/I コンパイラーの間には相違点があるため、PLI-ENTERPRISE と PLI-OTHER の 2 つの言語オプションがサポートされます。

PL/I にスキーマ・エレメント名をマップする場合の規則

CICS 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 PL/I 変数の固有で有効な名前を生成します。

1. A から Z、a から z、0 から 9、@、#、_、または $ 以外の文字は、「X」で置き換えられます。

   例えば、 monthly-total は monthlyXtotal となります。

   MAPPING-OVERRIDES パラメーターを使用して、他の文字の処理方法を変更できます。 例えば、値 **HYPHENS-AS-UNDERSCORES** を設定すると、JSON スキーマに含まれるハイフンは X ではなく下線に変換されます。例えば、monthly-total は monthly_total となります。
2. 同じスコープ内の重複した名前は、名前の 2 番目以降のインスタンスに 1 つまたは 2 つの数字を追加することによって固有にします。

   例えば、 year の 3 つのインスタンスは year 、 year1 、および year2 になります。

   もし上記の動作が望ましくないなら、ユーザーはユーティリティーの入力として MAPPING-OVERRIDES=NO-ARRAY-NAME-INDEXING を指定できます。これにより、名前の 2 番目以降のインスタンスへの 1 つまたは 2 つの数字の追加が無効になります。
3. JSON スキーマは、"type" 値が "array" で、キーワード "minItems" と "maxItems" が省略されるか、それぞれ別の値が指定されている場合には、変数の基数が可変になるように指定します。変数の基数が可変になるようスキーマで指定されている場合、フィールド名は接尾部 "_cont" および "_num" が付けられて作成されます。

   詳しくは、 324 ページの『DFHJS2LS 内のエレメントの可変配列』を参照してください。
4. JSON スキーマは、"required" キーワード配列 (周りを囲む JSON スキーマ "object" タイプに関連付けられている) に指定されていない変数がオプションとなるように指定します。オプション・フィールドの場合、追加のフィールドは、エレメント名に接尾部 _num が追加されて生成されます。実行時にこれは、JSON データに値が存在しなかったことを示す場合はゼロ、JSON データに値が存在した場合は非ゼロになります。
5. フィールド名は 31 文字までに制限されています。接頭部および接尾部を含めて生成された名前がこの長さを超えると、エレメント名が切り捨てられます。

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。

PL/I にスキーマ・タイプをマップする場合の規則

DFHJS2LS は、次の表に従ってスキーマ・タイプ値を PL/I データ・タイプにマップします。以下の点にも注意してください。

- **MAPPING-LEVEL** パラメーターを 1.2 以上に設定して、**CHAR-VARYING** パラメーターを NULL に設定すると、可変長文字データはヌル終了ストリングにマップされ、ヌル終止符として追加された 1 つの文字が割り振られます。
- **MAPPING-LEVEL** パラメーターを 1.2 以上に設定して、**CHAR-VARYING** パラメーターを指定しないと、可変長文字データはデフォルトでは、Enterprise PL/I の場合は VARYINGZ データ・タイプにマップされ、その他の PL/I の場合は VARYING データ・タイプにマップされます。
- 可変長バイナリー・データは、32 768 バイトより少ない場合は VARYING データ・タイプにマップされ、32 768 バイトを超える場合はコンテナーにマップされます。

JSON スキーマ・キーワード	**PL/I** のデータ記述

以下のすべて:
`"type":"null"`
`"type": []`
	サポートされていない

`"$schema":"http://json-schema.org/draft-04/schema#"`	このキーワードは無視されますが、ドラフト 04 JSON スキーマ仕様と互換性があることが想定されています。
`"title": "same text"` `"description": "more text"`	これらのキーワードは無視されます。
`"format": "<predefined values>"`	"format" キーワードは、生成された構造またはランタイム値のいずれかを変更するために使用されます。サポートされる "format" の使用方法については、以下を参照してください。

`"type":`
`"array",`
`"items": {<JSON Sub-schema>},`
`"additionalItems": false,`
`"maxItems":`
m
`,`
`"minItems":`
n
	多次元配列は、マッピング・レベル 4.3 以上でサポートされ、混合タイプ配列はサポートされません。
	
	"additionalItems" は、false になることが想定されています。他にサポートされている値はありません。
	
	"minItems" と "maxItems" の両方が存在し、それらが等しい場合、配列は固定基数として処理されます。それ以外の場合、配列は可変基数として処理されます。 324 ページの『DFHJS2LS 内のエレメントの可変配列』を参照してください。

JSON スキーマ・キーワード	**PL/I** のデータ記述
"type": "array", "uniqueItems": true	"uniqueItems" は、JSON 配列でサポートされていません。 <JSON Sub-schema> はサポートされている "type" を定義する必要がありますが、 "type" を "array" にすることはできません。 これは、生成される言語構造に対する制約事項です。
"type": "object", "additionalProperties": false, "properties": { ["<element name>": {<JSON Sub-schema>} [,]]* } "required": [["<element name>" [,]]*]	現在サポートされている JSON オブジェクトの唯一の形式は、名前付きエレメントの固定セットです。 これは、エレメント名を使用して構造 (または副構造) を生成します。 "additionalProperties" は、値が **ADDITIONAL-PROPERTIES-DEFAULT** パラメーターで設定されていない場合、false と想定されます。 "required" 配列にないか、"required" 配列が存在しない場合、"properties" オブジェクト内のエレメントはすべて "optional" と見なされます。 "optional" エレメントにはゼロから X までの可変オーディナリティーが提供されます (X は 1 または配列の項目の最大数、その項目は配列として定義)。 324 ページの『DFHJS2LS 内のエレメントの可変配列』を参照してください。

JSON スキーマ・キーワード	**PL/I** のデータ記述
`"type": "object",` `"additionalProperties": true`	オブジェクトは前の例のようにマップされ、追加プロパティーをサポートする追加フィールドがあります。 `"additionalProperties"` プロパティーは、**ADDITIONAL-PROPERTIES-DEFAULT** パラメーターで設定されていない場合、false と想定されます。 有効な場合、**ADDITIONAL-PROPERTIES-MAX** パラメーターで指定された値までスペースが割り当てられます。各スペースの文字数は、**ADDITIONAL-PROPERTIES-SIZE** パラメーターで設定されています。個々のプロパティーは、`PIC X(z)` フィールドにマップされます。*z* は、**ADDITIONAL-PROPERTIES-SIZE** パラメーターによって定義されます。 **ADDITIONAL-PROPERTIES-MAX** の値が 0 より大きい場合、プロパティーは、**maxItems** が設定された配列のプロパティーと同じようにマップされます。 **注:** CICS での z/OS Connect の使用など、CICS での JSON サポートを構成する方法はいくつかあります。古い CICS Java パイプライン・テクノロジーを使用する場合、**追加プロパティー**は、`com.ibm.cics.json.enableAxis2Handlers` JVM システム・プロパティーが true に設定されていないときにのみサポートされます。

`"type": "object",`
`"maxProperties": m,`
`"minProperties": n,`
`"patternProperties": {},`
`"dependencies":`
	これらのキーワードは、いずれも JSON オブジェクトではサポートされません。

`"type":"string"`
`"maxLength":`
m
`"pattern": "regular expression>",`
`"minLength":`
l
	`char[ z ]`
	
	ここで *z* の値は *m* に基づいていますが、 **CHAR-VARYING** パラメーターの設定に依存します。
	
	m は `"maxLength"` キーワードに基づき、固定長ストリングとして扱われます。
	
	`"pattern"` および `"minLength"` の制約事項は、コメントとしてのみ言語構造に渡されます。

JSON スキーマ・キーワード	**PL/I** のデータ記述
"type":"string" "maxLength": *m*	マッピング・レベル 4.0 以上で CCSID=1200 の場合: WIDECHAR(*z*) ここで *z* の値は *m* に基づいていますが、 **CHAR-VARYING** パラメーターの設定に依存します。 *m* は "maxLength" キーワードに基づき、固定長ストリングとして扱われます。

"*name*":{
"type":"string",
"format":"date-time"
}
	FIXED DECIMAL (15,0)
	
	DATETIME=PACKED15 の場合、すべてサポートされます。

"*name*":{
"type":"string",
"format":"uri"
}
	CHAR(*m*)
	
	ここで、m は "maxLength" キーワードに基づき、固定長ストリングとして扱われます。
	
	マッピング・レベル 4.0 以上で **CCSID=1200** の場合:
	
	WIDECHAR(*m*)
	
	ここで、*m* は "maxLength" キーワードに基づき、固定長ストリングとして扱われます。

"*name*":{
"type":"string",
"format":"base64Binary"
}
	CHAR(*m*)
	
	ここで、m は "maxLength" キーワードに基づきます。

"*name*":{
"type":"string",
"format":"hexBinary"
}
	CHAR(*m*)
	
	ここで、m は "maxLength" キーワードに基づきます。

JSON スキーマ・キーワード	**PL/I** のデータ記述

"*name*":{
"type":"string",
"format":"<predefined>"
}
	CHAR(*m*) ここで、*m* は "maxLength" キーワードに基づき、固定長ストリングとして扱われます。<predefined> は、*email*、*hostname*、*ipv4*、または *ipv6* のいずれかです。 関連する "pattern" がコメントに渡されます。
	
	マッピング・レベル 4.0 以上で **CCSID=1200** の場合:
	
	WIDECHAR(*m*)
	
	ここで、 *m* は "maxLength" キーワードに基づき固定長ストリングとして扱われます。 また、 <predefined> は *email*、*hostname*、 *ipv4*、 または *ipv6* のいずれかです。関連する "pattern" が使用され、コメントに渡されます。

"type":"boolean"	マッピング・レベル 1.1 以下: **Enterprise PL/I** SIGNED FIXED BINARY (7) その他の **PL/I** FIXED BINARY (7) マッピング・レベル 1.2 以上の場合 **Enterprise PL/I** BIT(7) BIT(1) その他の **PL/I** BIT(7) BIT(1) ここで、BIT(7) は位置合わせのために提供されており、BIT(1) にはブールでマップされた値が含まれます。
"type": "integer", "exclusiveMaximum": true, "exclusiveMinimum": true, "multipleOf": n	"exclusiveMaximum" および "exclusiveMinimum" の制約事項は、コメントとしてのみ言語構造に渡されます。 "multipleOf" は無視されます。
"type":"integer", minimum:-128, maximum:127	**Enterprise PL/I** SIGNED FIXED BINARY (7) その他の **PL/I** FIXED BINARY (7)

JSON スキーマ・キーワード	**PL/I** のデータ記述
"type":"integer", minimum:0, maximum:255	**Enterprise PL/I** UNSIGNED FIXED BINARY (8) その他の **PL/I** FIXED BINARY (8)
"type":"integer", minimum:-32768, maximum:32767	**Enterprise PL/I** SIGNED FIXED BINARY (15) その他の **PL/I** FIXED BINARY (15)
"type":"integer", minimum:0, maximum:65535	**Enterprise PL/I** UNSIGNED FIXED BINARY (16) その他の **PL/I** FIXED BINARY (16)
"type":"integer", minimum:-2147483648, maximum:2147483647	**Enterprise PL/I** SIGNED FIXED BINARY (31) その他の **PL/I** FIXED BINARY (31)
"type":"integer", minimum:0, maximum:4294967295	マッピング・レベル 1.1 以下: **Enterprise PL/I** UNSIGNED FIXED BINARY(32) マッピング・レベル 1.2 以上の場合 **Enterprise PL/I** CHAR(*y*) ここで *y* は 16 MB より小さい固定長です。 すべてのマッピング・レベル: その他の **PL/I** BIT(64)

JSON スキーマ・キーワード	**PL/I** のデータ記述

`"type":"integer",`
`minimum:-9223372036854775808,`
`maximum:9223372036854775807`
	マッピング・レベル 1.1 以下:
	Enterprise PL/I
	`SIGNED FIXED BINARY(63)`
	注: LIMITS コンパイラー・ディレクティブは、PL/I コンパイラーによるこのフィールドの解釈方法に影響する場合があります。CICS は、このフィールドが宣言されたサイズであることを想定しますが、コンパイラーはこのフィールドをより小さなスペースに最適化することがあるため、不一致が生じる可能性があります。このような問題を回避するには、LIMITS(FIXEDBIN(63)) コンパイル時オプションを使用します。
	マッピング・レベル 1.2 以上の場合
	Enterprise PL/I
	`CHAR(` *y* `)`
	ここで *y* は 16 MB より小さい固定長です。
	すべてのマッピング・レベル:
	その他の **PL/I**
	`BIT(64)`

JSON スキーマ・キーワード	**PL/I** のデータ記述

`"type":"integer",`
`minimum:0,`
`maximum:18446744073709551615`
	マッピング・レベル 1.1 以下:
	Enterprise PL/I
	`UNSIGNED FIXED BINARY(64)`
	注: LIMITS コンパイラー・ディレクティブは、PL/I コンパイラーによるこのフィールドの解釈方法に影響する場合があります。CICS は、このフィールドが宣言されたサイズであることを想定しますが、コンパイラーはこのフィールドをより小さなスペースに最適化することがあるため、不一致が生じる可能性があります。このような問題を回避するには、LIMITS(FIXEDBIN(63)) コンパイル時オプションを使用します。
	マッピング・レベル 1.2 以上の場合
	Enterprise PL/I
	`CHAR(` *y* `)`
	ここで *y* は 16 MB より小さい固定長です。
	その他の **PL/I**
	`BIT(64)`

`"type":"number"` `"description":"decimal"`	`FIXED DECIMAL(` *n* `,` *m* `)`

JSON スキーマ・キーワード	**PL/I** のデータ記述

`"type":"number"`
`"description":"float"`
	マッピング・レベル 1.0 および 1.1 の場合
	• `CHAR(32)`
	マッピング・レベル 1.2 以上の場合
	Enterprise PL/I
	`DECIMAL FLOAT(6) HEXADEC`
	その他の **PL/I**
	`DECIMAL FLOAT(6)`
	注: IBM Hexadecimal Floating Point (HFP) データ表記は、JSON に使用される IEEE-754-1985 表記とまったく同じではありません。一部の値が、ある表記から別の表記に正確に変換されない可能性があります。極端に大きいまたは小さい値は、float データ型に対して有効ではない場合があります。 HFP 表記との間の変換時に、一部の値で精度が失われる可能性があります。正確な変換が重要になる場合、DECIMAL FLOAT データ型を使用せずに、固定精度のデータ型を代わりに使用することを検討してください。

`"type":"number"`
`"description":"double"`
	マッピング・レベル 1.0 および 1.1 の場合
	• `CHAR(32)`
	マッピング・レベル 1.2 以上の場合
	Enterprise PL/I
	`DECIMAL FLOAT(16) HEXADEC`
	その他の **PL/I**
	`DECIMAL FLOAT(16)`
	注: IBM Hexadecimal Floating Point (HFP) データ表記は、JSON に使用される IEEE-754-1985 表記とまったく同じではありません。一部の値が、ある表記から別の表記に正確に変換されない可能性があります。極端に大きいまたは小さい値は、float データ型に対して有効ではない場合があります。 HFP 表記との間の変換時に、一部の値で精度が失われる可能性があります。正確な変換が重要になる場合、DECIMAL FLOAT データ型を使用せずに、固定精度のデータ型を代わりに使用することを検討してください。

JSON スキーマ・キーワード	**PL/I** のデータ記述

`oneOf: [ { "properties": { "A": {...} } } ,`
`  { "properties": { "B": {...} } } ]`
`anyOf: [ { "properties": { "A": {...} } } ,`
`  { "properties": { "B": {...} } } ]`
`allOf: [ { "properties": { "A": {...} } } ,`
`  { "properties": { "B": {...} } } ]`
	オプションの配列を使用した論理パスは、`{ "properties": { "A":..., "B":... } }` というタイプの単一オブジェクトが定義されている場合と同様にマージされます。JSON プロパティーはそれぞれオプションとして扱われます。同じプロパティー名の競合する定義がサブオプションに含まれている場合は、エラー・メッセージが発行されます。例えば、単一プロパティーがストリング (パス A 内) とオブジェクト (パス B 内) の両方として定義されている場合、そのような定義はサポートされず、エラー・メッセージが表示されます。
	
	JSON スキーマで複雑な論理構造を定義できますが、複雑な論理構造内で暗黙的に指定された細かい点は、言語構造へのマッピングで失われる可能性があります。変換プロセスは、スキーマからの組み合わせルールを強制しようとしません。指定された JSON プロパティーが存在するか存在しないかを示す言語構造フィールドとのみ対話します。
	
	サブオプションに同じプロパティー名で互換性のある定義が含まれている場合、DFHJS2LS は、制約の関連パターンをマージしようとしますが、プロセスではいくつかの細かい点が失われる場合があります。例えば、以下の定義を想定します。
	
	`"A": { "oneOf": [ { "type": "string", "maxLength": 5}, {"type": "string", "minLength": 7, "maxLength": 8} ] }`
	
	『A』 は、最大 5 文字の長さのストリング、または長さが 7 から 8 文字のストリングのいずれかとして定義されています。構成のマージにより、6 文字の長さが無効であることは認識されず、0 から 8 文字のストリングにマッピングされる可能性があります。
	
	論理構成を含むほとんどのシナリオでは単純な言語構造にマップされますが、複雑な論理構成では、マッピング処理中に妥協が図られることがあります。最良の結果を得るために、JSON スキーマで論理構成を使用せずに、同じ JSON プロパティーに代替宣言を定義してください。

注: CICS では、signed long の最大値 (2^63 - 1) より大きい整数値を変換できません。ただし、整数値が引用符で囲まれている場合は変換できます。

注: 数値型のスキーマに指定された最小値および最大値は、PL/I データ型にマップされる目的でのみ使用されます。実行時にデータがこれらの値に基づいて検証されることはありません。

DFHJS2LS 内のエレメントの可変配列

JSON には、エレメント数が変化する配列を含めることができます。一般に、エレメント数が変化する JSON スキーマは 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。 CICS はコンテナー・ベースのマッピングまたはインライン・マッピングを使用して JSON データにおけるエレメント数の変化に対応します。

エレメント数が変化する配列を JSON スキーマ内で表現するには、"type" 値が "array" のスキーマで、キーワード minItems と maxItems を使用します。

- minItems キーワードは、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。デフォルトは、値 0 です。
- maxItems キーワードは、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、minItems キーワードの値以上の任意の正の整数値を指定できます。
- maxItems キーワードがない場合、配列が無制限であることを示します。

可変配列 "maxItems":1 によってオプション・フィールドを表すことができます。例えば、"component" という名前の 8 バイトのオプション・ストリングは次のようになります。

```
"properties":{
"component": {
"type":"array",
"maxItems":1,
"items": {
"type": "string",
"maxLength": 8
}
}
},
"required": ["component"]
```

次のように、"required" キーワード値にフィールド名を含めないことにより、同じ効果を得ることができます。

```
"properties":{
"component": {
"type": "string",
"maxLength": 8
}
}
```

一般に、エレメント数が変化する JSON スキーマは 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。 このような場合に対処するため、CICS

は一連のコンテナーとしてアプリケーション・プログラム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。 これらの構造はアプリケーションへの入出力として使用されます。

- CICS が JSON データをアプリケーション・データに変換する場合、これらの構造にアプリケーション・データが追加され、アプリケーションがそのデータを読み取ります。
- CICS がアプリケーション・データを JSON データに変換する場合、アプリケーションによって構造内に追加されたアプリケーション・データが読み取られます。

以下の例は、こうしたデータ構造のフォーマットを示しています。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。

例 1. 固定のエレメント数

この例では、ちょうど 3 回発生するエレメントを示しています。

```
"properties":{
"component": {
"type": "array",
"maxItems": 3,
"minItems": 3,
"items": {
"type": "string",
"maxLength": 8
}
}
},
"required": ["component"]
```

この例では、エレメントが出現回数が事前に分かっているので、単純な COBOL 宣言、または他の言語のこれに相当するもので固定長配列として表すことができます。

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

例 2. マッピング・レベル 2 以下における変化するエレメント数

この例では、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。

```
"properties":{
"component": {
"type": "array",
"maxItems": 5,
"minItems": 1,
"items": {
"type": "string",
"maxLength": 8
}
}
},
"required": ["component"]
```

主データ構造には、2 つのフィールドの宣言が格納されます。 CICS が JSON データをバイナリー・データに変換すると、最初のフィールドである component-num に

はエレメントがJSON データ内に現れる回数が格納され、2 番目のフィールドである component-cont には、コンテナーの名前が格納されます。

```
05 component-num PIC S9(9) COMP-5
05 component-cont PIC X(16)
```

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 DFHJS-component
02 component PIC X(8)
```

エレメントの出現回数を確認するには、component-num (1 から 5 の範囲の値が含まれる) の値を調べる必要があります。エレメント・コンテンツは component-cont で指定されたコンテナー内にあります。このコンテナーはエレメントの配列を保持しており、各エレメントは DFHJS-component データ構造によってマップされています。

minItems="0" (または未指定) で、かつ maxItems="1" の場合、このエレメントはオプションです。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、以下のように component-num 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 エレメントはなく、component-cont の内容は未定義です。
- この値が 1 の場合、component エレメントは component-cont 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、DFHJS-component データ構造によってマップされます。

注: JSON データが単一の繰り返しエレメントで構成される場合、DFHJS2LS は 2 つの言語構造を生成します。 主要な言語構造は、配列エレメントの数と、エレメントの配列を保持するコンテナーの名前を含んでいます。 2 番目の言語構造は、繰り返しエレメントの単一インスタンスをマップします。

例 3 マッピング・レベル 2.1 以上における変化するエレメント数

マッピング・レベル 2.1 では、CICS 支援機能で **INLINE-MAXOCCURS-LIMIT** パラメーターを使用できます。 **INLINE-MAXOCCURS-LIMIT** パラメーターは変化するエレメント数を処理する方法を指定します。 変化するエレメント数のマッピング・オプションは、 325 ページの『例 2. マッピング・レベル 2 以下における変化するエレメント数』に記述されているコンテナー・ベースのマッピング、またはインライン・マッピングです。 このパラメーターの値は、0 - 32767 までの範囲の正整数です。

- **INLINE-MAXOCCURS-LIMIT** のデフォルト値は 1 であり、オプション・エレメントがインラインで確実にマップされます。
- **INLINE-MAXOCCURS-LIMIT** パラメーターに値 0 が設定されている場合、インライン・マッピングは使用されません。
- maxItems が **INLINE-MAXOCCURS-LIMIT** の値以下の場合、インライン・マッピングが使用されます。
- maxItems が **INLINE-MAXOCCURS-LIMIT** の値以上の場合、コンテナー・ベースのマッピングが使用されます。

変化するエレメント数をインラインでマップすると、配列 (上記の例では発生するオカレンスが固定) およびカウンターが生成されます。 component-num フィールドは存在するエレメントのインスタンス数を示し、それらのインスタンスは配列によ

って示されます。 325 ページの『例 2. マッピング・レベル 2 以下における変化するエレメント数』で示される例では、**INLINE-MAXOCCURS-LIMIT** が 5 以下の場合に生成されるデータ構造は次のようになります。

```
05 component-num PIC S9(9) COMP-5 SYNC.
05 component OCCURS 5 PIC X(8).
```

最初のフィールド (component-num) は前のセクションで示されたコンテナー・ベース・マッピングの例の出力と同じです。 2 番目のフィールドには長さが 5 の配列があり、生成される可能性のある最大エレメント数を収容できる大きさです。

インライン・マッピングは、エレメントの出現回数およびデータが収容されるコンテナーの名前を格納するコンテナー・ベースのマッピングとは異なり、現行のコンテナーにあるすべてのデータを格納します。 現行のコンテナーにデータを格納するとパフォーマンスが向上するので、インライン・マッピングの方が一般的には望ましいといえます。

例 4. ネストされた変数配列

複雑な JSON スキーマには可変の繰り返しエレメントを含めることができ、そのエレメントにはさらに可変の繰り返しエレメントを含めることが可能です。この場合、記述される構造は、例で説明した 2 つのレベルを超えます。

この例は、1 回から 5 回出現する可能性がある必須エレメント "component1" 内でネストされたオプションのエレメント "component2" を示しています。

```
"properties":{
"component1": {
"type": "array",
"maxItems": 5,
"minItems": 1,
"items": {
"type": "object",
"properties":{
"component2":{
"type": "string",
"maxLength": 8
}
},
"required": ["component2"]
}
},
"required": ["component1"]
```

最上位のデータ構造は、前の例のデータ構造とまったく同じです。

```
05 component1-num PIC S9(9) COMP-5
05 component1-cont PIC X(16)
```

ただし、2 番目のデータ構造には、以下のエレメントが格納されます。

```
01 DFHJS-component1
02 component2-num PIC S9(9) COMP-5
02 component2-cont PIC X(16)
```

3 番目の構造には以下のエレメントが格納されます。

```
01 DFHJS-component2
02 component2 PIC X(8)
```

最外部エレメント "component1" の出現回数は、component1-num に格納されます。

component1-cont で指定されたコンテナーには、2 番目のデータ構造 (DFHJS-component1) のその数のインスタンスを持つ配列が含まれています。

component2-cont の各インスタンスは、異なるコンテナーを指定します。それぞれ、第 3 レベルの構造 (DFHJS-component2) によってマップされたデータ構造が含まれています。

この構造を説明するために、例と一致する次のような JSON データのフラグメントについて考えてみます。

```
{"component1":
[
{
"component2": "string1"
},
{
"component2": "string2"
},
]
}
```

"component1" は 3 回発生します。 最初の 2 つには、"component2" のインスタンスが含まれていますが、3 番目のインスタンスには含まれていません。

最上位のデータ構造では、component1-num に値 3 が含まれています。 component1-cont で指定されたコンテナーには、DFHJS-component1 の次の 3 つのインスタンスがあります。

1. 第 1 のインスタンスでは、component2-num の値は 1 で、component2-cont で指定されたコンテナーは *string1* を保持します。
2. 第 2 のインスタンスでは、component2-num の値は 1 で、component2-cont で指定されたコンテナーは *string2* を保持します。
3. 第 3 のインスタンスでは、component2-num の値は 0 で、component2-cont の内容は未定義です。

このインスタンスでは、完全なデータ構造は、次の合計 4 つのコンテナーによって表現されます。

- コンテナー DFHJS-DATA 内のルート・データ構造。
- component1-cont で指定されたコンテナー。
- component2-cont の最初の 2 つのインスタンスで指定された 2 つのコンテナー。

オプションの構造と required キーワード

データ構造は、JSON スキーマ "type" の "object" で定義します。"properties" キーワードで指定したオブジェクトを使用して、スキーマによりフィールド名が個々のタイプと関連付けられます。これらのフィールドが、JSON スキーマによって記述された JSON データの一部となることが必要かどうかは、"required" キーワードで指定される配列により制御されます。この配列には、JSON データ内に存在する必要があるすべてのフィールド名がリストされます。したがって、オプションのフィールドを表すには、そのフィールドをこの配列から除きます。あるいは、配列を空にはできないので、"required" キーワードを除きます。後者の場合、すべてのフィールドがオプションとなります。

オプションのフィールドは、0 個または 1 個のエレメントから成る変数配列として扱われます。これは、エレメント名に接尾部 "-num" が付加されたフィールドを追加します。全長が 28 文字を超える場合、エレメント名が切り捨てられます。これは実行時に、JSON データに値が存在していたことを示す場合は非ゼロ、そうでない場合はゼロになります。

以下の例に 2 つのフィールドを示します。一方は "required-structure" という必須フィールドで、もう一方は "optional-structure" というオプション・フィールドです。

```
{
"type": "object",
"properties": {
"required-structure": {
"type":"string",
"maxLength": 8
},
"optional-structure": {
"type":"string",
"maxLength": 8
}
},
"required": [
"required-structure"
]
}
```

生成される COBOL 構造には両方のフィールドが示されますが、2 番目の前にある "optional-structure-num" はエレメントに関する整数カウントで、0 は「なし」、1 は存在することを表しています。この値は、"optional-structure" に有効なデータが含まれるかどうかを示すように設定されます。

```
03 OutputData.
06 required-structure PIC X(8).
06 optional-structure-num PIC S9(9) COMP-5 SYNC.
06 optional-structure PIC X(8).
```

アプリケーション・データでの UTF-16 サポート

CICS Web サービスでは、UTF-16 でエンコードされたアプリケーション・データから XML または JSON への変換と、XML または JSON から UTF-16 エンコード・アプリケーション・データへの変換がサポートされます。複数の言語でデータを保管および処理する必要がある場合には UTF-16 を使用してください。

CICS SOAP および JSON Web サービスでは、UTF-16 でエンコードされたアプリケーション・データから XML または JSON への変換と、XML または JSON から UTF-16 でエンコードされたアプリケーション・データへの変換がサポートされます。可変幅エンコード・スキームである Unicode を使用すると、システムは効率的にデータを扱うことができます。

UTF-16 は Unicode の可変幅エンコード方式であり、各文字が 2 バイトまたは 4 バイトで表されます。 CICS Web サービスはアプリケーション・データ用に CCSID 1200 をサポートします。これは、IBM 専用領域を含む UTF-16 BE (ビッグ・エンディアン) です。この動作は、サポートされるすべての言語で UTF-16 サポートと整合しています。

UTF-16 はマッピング・レベル 4.0 以上でサポートされます。アシスタントでマッピング設定を使用して、アプリケーション・データの変換方法をカスタマイズすることができます。 XML マッピング・レベルについて詳しくは、CICS アシスタントのマッピング・レベルを参照してください。JSON マッピング・レベルについて詳しくは、CICS JSON アシスタントのマッピング・レベルを参照してください。

注: UTF-16 は EBCDIC エンコード方式に比べてより多くの処理時間を必要とし、ストレージ効率が低くなります。さらに、複数の種類のエンコード方式を混合すると、実行時に追加の処理が発生します。

XML または JSON スキーマから言語構造への UTF-16 のマッピング

UTF-16 のサポートは、Web サービスの作成方法に応じて異なります。 XML または JSON スキーマから言語構造へのマッピング (トップダウン・マッピングともいう) には、次のような特性があります。 UTF-16 が有効になっている場合、すべてのテキスト・フィールドは UTF-16 フィールドにマップされ、COBOL での数値表示データ型は EBCDIC としてマップされます。 UTF-16 を使用するには、DFHJS2LS、DFHSC2LS、または DFHWS2LS の CCSID パラメーターを 1200 に設定してください。

例えば、WSDL の中に以下の XML スキーマ断片が存在する場合、

```
<xsd:element name="myString" nillable="false">
<xsd:simpleType>
<xsd:restriction base="xsd:string">
<xsd:maxLength value="20"/>
</xsd:restriction>
</xsd:simpleType>
</xsd:element>
```

DFHWS2LS アシスタントは COBOL 言語構造で以下のフィールドを生成します。

```
myString PIC N(
20
) USAGE NATIONAL
```

Web サービス・アシスタントの CHAR-MULTIPLIER パラメーターを使用すると、アシスタントによって生成されるフィールドの長さを指定できます。

CHAR-MULTIPLIER

UTF-16 を使用する場合、 **CHAR-MULTIPLIER** パラメーターに有効な値は、2 または 4 のみで、 2 はデフォルト値です。

CHAR-MULTIPLIER = 2。 ここで、スキーマは maxlength *x* のストリングを表し、 PIC N(*x*) を生成します。 **CHAR-MULTIPLIER**=2 を設定した場合、UTF-16 ストリングでの代理ペアは使用不可にはなりませんが、フィールドに入る文字数が影響を受けます。

CHAR-MULTIPLIER = 4 は、PIC N(*2x*) を生成します。 **CHAR-MULTIPLIER**=4 である場合、1 つのエンコード・ユニットで表現できる文字がストリングに含まれるならば、実行時の値が埋め込まれます。

言語構造から XML または JSON スキーマへの UTF-16 のマッピング

言語構造から XML または JSON スキーマへのマッピング (ボトムアップ・マッピングともいう) は、トップダウン・マッピングとは異なる方法で管理されます。言

語構造で UTF-16 ストリングが宣言されている場合、CICS はデータが UTF-16 でエンコードされていると解釈し、そうでない場合は EBCDIC エンコード方式のデータを想定します。 DFHLS2JS、DFHLS2SC、または DFHLS2WS の CCSID パラメーターは、アプリケーション・データ内の EBCDIC テキストのエンコードを示します (UTF-16 を示すようこれを設定してはなりません)。

UTF-16 文字として解釈されるデータ型は、COBOL では `PIC N (n)`、PL/I では `WIDECHAR(n)`、C および C++ では `char16_t[n]` です。

Web サービス・アシスタントの CHAR-USAGE パラメーターを使用して、データ型を指定できます。

CHAR-USAGE

COBOL では、各国語データ型 `PIC N` を UTF-16 または DBCS データ用に使用できます。この設定は `NSYMBOL` コンパイラー・オプションによって制御されます。データが適切に扱われるようにするには、アシスタントの **`CHAR-USAGE`** パラメーターを `NSYMBOL` コンパイラー・オプションと同じ値に設定する必要があります。 UTF-16 を使用する場合には、通常、これは `CHAR-USAGE=NATIONAL` に設定されます。

同じコピーブック内で UTF-16 と DBCS のデータを含む各国語データ型を混合する必要がある場合は、個々のフィールドで `USAGE NATIONAL` または `USAGE DISPLAY-1` 修飾子を使用できます。

注: DFHLS2WS、DFHLS2SC、および DFHLS2JS は COBOL の `GROUP USAGE NATIONAL` 節をサポートしません。

言語構造からのマッピングの生成

JavaScript Object Notation (JSON) をアプリケーション・データから作成したり、JSON からアプリケーション・データを作成するには、マッピングを作成して、CICS が実行時にデータおよび JSON をどのように変換するかを記述します。これは、どのアプリケーション・データ・レコードからでも開始できます。例えば、COMMAREA、VSAM ファイル、一時ストレージ・キュー、または IBM Db2 レコードから開始できます。

始める前に

マッピングを作成する前に、以下の前提条件が満たされていることを確認する必要があります。

- 区分データ・セット内に、アプリケーション・レコードを記述する言語構造が必要です。 言語構造は、COBOL、PL/I、C、C++ など、CICS JSON 支援機能でサポートされている高水準言語のいずれでも記述できます。
- DFHLS2JS が実行されて z/OS UNIX を使用するユーザー ID を構成する必要があります。
- このユーザー ID には、言語構造にアクセスするための読み取り権限と、z/OS UNIX の適切なディレクトリーに出力するための書き込み権限が必要です。
- このユーザー ID で Java を実行するためには、この ID に十分なストレージを割り振る必要があります。 サポートされている任意のバージョンの Java を使用できます。

このタスクについて

CICS JSON 支援機能を使用して、アプリケーション・レコードのデータ・マッピングを作成します。 CICS JSON 支援機能は、CICS バンドルを作成します。また、言語構造内で識別された非サポート項目があれば、エラー・メッセージを発行します。 CICS JSON 支援機能に関する参照情報では、各高水準言語に適用される制限がリストされています。

手順

DFHLS2JS バッチ・ジョブを実行します。DFHLS2JS には、特定のコード・ページの選択など、要件を満たすために選択するオプション・パラメーターが用意されています。 少なくとも、以下のパラメーターを使用します。

- 言語構造の高水準言語を **LANG** パラメーターで指定します。
- バンドル・リソースの名前と場所を **BUNDLE** パラメーターで指定します。
- マッピング・レベルを **MAPPING-LEVEL** パラメーターで指定します。どのマッピング・レベルでも使用できますが、最も高機能のマッピング・オプションを選択するには、最新のマッピング・レベルを使用します。
- **PDSMEM** パラメーターと **PDSCP** パラメーターで、アプリケーション・レコードを記述する言語構造の場所とコード・ページを指定します。
- `.json` JSON スキーマ・ファイルの名前と場所を **JSON-SCHEMA** パラメーターで指定します。このファイルが存在しない場合、DFHLS2JS は JSON スキーマを作成しますが、ディレクトリー構造は作成しません。
- JSONTRANSFRM バンドル・リソースに使用される名前を指定します。この名前は、JSON マッピングを識別するために、アプリケーションによって使用されます。

  バッチ・ジョブは、z/OS UNIX 上のバンドル・ディレクトリー構造を作成します。バンドル・ディレクトリーには、バンドル・マニフェストが含まれる META-INF サブディレクトリーがあります。 また、バッチ・ジョブは、**JSONTRANSFRM** および **JSON-SCHEMA** パラメーターに指定したファイル名を使用して、JSON スキーマおよび JSON バインディングをバンドルに作成します。

  この JSON バインディングを指定する BUNDLE リソースをインストールします。 JSONTRANSFRM バンドル・リソースにより、JSON スキーマとバインディング・ファイルの場所を定義する JSONTRANSFRM リソースが動的に作成されます。 CICS Explorer の「バンドル」操作ビューおよび「バンドル・パーツ」操作ビューを使用すると、インストールされた BUNDLE リソースの状況を確認できます。

タスクの結果

1 つの JSON 変換を含む CICS バンドルが生成されます。

以下の例は、パラメーターの最小セットが指定された DFHLS2JS を示しています。

```
//LS2JS JOB 'accounting information',name,MSGCLASS=A
// SET QT=''''
//JCLLIB JCLLIB ORDER=FPHLQ.SDFHMOBI
//JAVAPROG EXEC DFHLS2JS,
//INPUT.SYSUT1 DD *
```

```
LANG=COBOL
BUNDLE=/u/exampleapp/bundle/test1
LOGFILE=/u/exampleapp/jsbind/example.log
MAPPING-LEVEL=3.0
PDSLIB=//CICSHLQ.SDFHSAMP
PDSMEM=CPYBK2
JSONTRANSFRM=example.jsbind
JSON-SCHEMA=/u/exampleapp/example.json
/*
```

次のタスク

アプリケーション・データを JSON に変換したりその逆の変換を行ったりするためのアプリケーション・プログラムを作成します。 両方の変換に同じマッピングを使用できます。

JSON スキーマからのマッピングの生成

JavaScript Object Notation (JSON) からアプリケーション・データを作成したり、アプリケーション・データから JSON を作成するには、マッピングを作成して、CICS が実行時にデータおよび JSON をどのように変換するかを記述します。 JSON スキーマから開始できます。言語構造およびマッピングを生成した後、言語構造を使用して CICS アプリケーションを開発し、JSON をアプリケーション・データに変換できます (その逆にも変換できます)。

始める前に

マッピングを作成する前に、以下の前提条件が満たされていることを確認する必要があります。

- JSON レコードを記述する JSON スキーマが存在している必要があります。
- DFHJS2LS が実行されて z/OS UNIX を使用するユーザー ID を構成する必要があります。
- このユーザー ID には、JSON スキーマにアクセスするための読み取り権限と、z/OS UNIX の適切なディレクトリーに出力するための書き込み権限が必要です。
- このユーザー ID で Java を実行するためには、この ID に十分なストレージを割り振る必要があります。 サポートされている任意のバージョンの Java を使用できます。

このタスクについて

CICS JSON 支援機能を使用して、アプリケーション・レコードのデータ・マッピングを作成します。 CICS JSON 支援機能では、CICS バンドルが作成され、言語構造内で識別された非サポート項目がある場合は、エラー・メッセージが発行されます。 CICS JSON 支援機能に関する参照情報では、各高水準言語に適用される制限がリストされています。詳しくは、 463 ページの『DFHJS2LS: リンク可能インターフェースに関する JSON スキーマから高水準言語への変換』を参照してください。

手順

DFHJS2LS バッチ・ジョブを実行します。DFHJS2LS には、特定のコード・ページの選択など、要件を満たすために選択するオプション・パラメーターが用意されています。 少なくとも、以下のパラメーターを使用します。

- バンドル・リソースの名前と場所を **BUNDLE** パラメーターで指定します。
- JSON スキーマ・ファイルの名前と場所を **JSON-SCHEMA** パラメーターで指定します。
- マッピング・レベルを **MAPPING-LEVEL** パラメーターで指定します。どのマッピング・レベルでも使用できますが、最も高機能のマッピング・オプションを選択するには、最新のマッピング・レベルを使用します。
- 生成される言語構造の高水準言語を **LANG** パラメーターで指定します。
- **PDSMEM** パラメーターと **PDSCP** パラメーターで、アプリケーション・レコードを記述する言語構造の場所とコード・ページを指定します。DFHJS2LS は言語構造を作成しますが、ディレクトリー構造は作成しません。
- CICS で JSONTRANSFRM バンドル・リソースに使用される名前を指定します。この名前は、JSON マッピングを識別するために、アプリケーションによって使用されます。

  バッチ・ジョブは、z/OS UNIX 上のバンドル・ディレクトリー構造を作成します。バンドル・ディレクトリーには、バンドル・マニフェストが含まれる META-INF サブディレクトリーがあります。 また、バッチ・ジョブは、**JSONTRANSFRM** および **JSON-SCHEMA** パラメーターに指定したファイル名を使用して、JSON バインディングを作成し、JSON スキーマをバンドル・ディレクトリーにコピーします。さらに、バッチ・ジョブは **PDSMEM** および **PDSLIB** パラメーターで指定した場所に言語構成も作成します。

  この JSON バインディングを指定する BUNDLE リソースをインストールします。 JSONTRANSFRM バンドル・リソースにより、JSON スキーマとバインディング・ファイルの場所を定義する JSONTRANSFRM リソースが動的に作成されます。このバンドル・リソースは、インストールしたバンドルの内容を CICS Explorer を使用して表示するときに CICS に表示されます。これは、通常の CICS リソースではないため、CEMT または CICSPlex SM WUI を使用するときには表示されません。

タスクの結果

1 つの JSON 変換を含む CICS バンドルが生成されます。言語構造が生成されます。

以下の例は、パラメーターの最小セットが指定された DFHJS2LS を示しています。

```
//JS2LS JOB 'accounting information',name,MSGCLASS=A
// SET QT=''''
//JCLLIB JCLLIB ORDER=FPHLQ.SDFHMOBI
//JAVAPROG EXEC DFHJS2LS,
//INPUT.SYSUT1 DD *
LANG=COBOL
BUNDLE=/u/exampleapp/bundle/test2
LOGFILE=/u/exampleapp/jsbind/example.log
MAPPING-LEVEL=3.0
PDSLIB=/u/exampleapp
```

```
PDSMEM=CPYBK2
JSONTRANSFRM=example.jsbind
JSON-SCHEMA=/u/exampleapp/example.json
/*
```

次のタスク

アプリケーション・データを JSON に変換したりその逆の変換を行ったりするためのアプリケーション・プログラムを作成します。 両方の変換に同じマッピングを使用できます。

DFHJSON へのリンクによるアプリケーション・データの JSON への変換

JSON 変換プログラムにリンク可能なインターフェースである DFHJSON は、 アプリケーション・データと JSON 間で変換を行うために呼び出すことができる CICS 提供プログラムです。アプリケーション・プログラムでは、DFHJSON にリンクすることによって、アプリケーション・データを JSON に変換できます。

始める前に

JSON バインディングと JSON スキーマを定義する、有効な JSONTRANSFRM バンドル・リソースが必要です。Java を使用して変換を実行しようとする場合、Axis2 JVM サーバーが既に実行されている必要があります。

このタスクについて

アプリケーション・プログラムを作成または更新して、変換を実行するため、CICS 提供プログラム DFHJSON にリンクします。

手順

1. アプリケーション・プログラムはチャネル (例えば、*MyChannelName*) を作成し、以下のコンテナーをそのチャネルに入れる必要があります。
   - DFHJSON-DATA
   - DFHJSON-TRANSFRM
   - DFHJSON-JVMSERVR (オプション)

   これらのコンテナーについて詳しくは、JSON 変換プログラム・リンク可能インターフェース・コンテナーを参照してください。
2. 以下の **EXEC CICS LINK PROGRAM** コマンドを使用して、データを JSON に変換します。

   ```
   EXEC CICS LINK PROGRAM('DFHJSON') CHANNEL('MyChannelName')
   ```
3. コンテナー DFHJSON-ERROR を取得し、変換中にエラーが発生したかどうかを確認します。
4. コンテナー DFHJSON-JSON を取得し、アプリケーションで JSON を利用します。
5. アプリケーションをインストールします。

タスクの結果

アプリケーションが **LINK** コマンドを実行すると、CICS は JSONTRANSFRM バンドル・リソースをチェックして JSON バインディングのマッピングを検索し、チャネル上のコンテナーを使用してアプリケーション・バイナリー・データを JSON に変換します。 JSON は、返されるときに、DFHJSON-JSON コンテナーに配置されます。この JSON は、JSONTRANSFRM バンドル・リソースで定義されている JSON スキーマに準拠しています。

次のタスク

同じマッピングを使用して JSON をアプリケーション・データに変換することもできます。 詳しくは、 530 ページの『DFHJSON へのリンクによる JSON のアプリケーション・データへの変換』を参照してください。

TRANSFORM DATATOJSON API コマンドを使用した、アプリケーション・データの JSON への変換

アプリケーション内で **TRANSFORM DATATOJSON** API コマンドを使用することで、アプリケーション・データを JSON に変換できます。

始める前に

JSON バインディングと JSON スキーマを定義する、有効な JSONTRANSFRM リソースが必要です。Java を使用して変換を実行しようとする場合、Axis2 JVM サーバーが既に実行されている必要があります。

このタスクについて

そのアプリケーションはチャネル・ベースのインターフェースを使用する必要があります。

手順

1. チャネルを作成して、変換対象のアプリケーション・データが含まれる入力コンテナーをそのチャネルに入れます。

   注: **TRANSFORM DATATOJSON** コマンドが完了すると、JSON 出力が含まれる出力コンテナーもこのチャネルに入れられます。**TRANSFORM DATATOJSON** を発行する前に、出力コンテナーを作成しないでください。このコンテナーは、このコマンドの一環として作成されて、データが取り込まれます。

2. **TRANSFORM DATATOJSON** コマンドを使用して、データを JSON に変換します。以下に例を示します。

```
EXEC CICS TRANSFORM DATATOJSON CHANNEL(
ChannelName
) INCONTAINER(
InpContainerName
) OUTCONTAINER(
OutContainerName
) TRANSFORMER(
BundleName
)
```

タスクの結果

アプリケーションが **TRANSFORM DATATOJSON** コマンドを実行すると、CICS は JSONTRANSFRM バンドル・リソースをチェックして JSON バインディングのマッピングを検索し、チャネル上のコンテナーを使用してアプリケーション・バイナリー・データを JSON に変換します。この JSON が、**TRANSFORM DATATOJSON** コマンドの **OUTCONTAINER** オプションで指定したコンテナーに格納されて返されます。このオプションを省略すると、デフォルトで DFHJSON-JSON が使用されます。この JSON は、JSONTRANSFRM バンドル・リソースで定義されている JSON スキーマに準拠しています。

DFHJSON へのリンクによる JSON のアプリケーション・データへの変換

JSON 変換プログラムにリンク可能なインターフェースである DFHJSON は、 アプリケーション・データと JSON 間で変換を行うために呼び出すことができる CICS 提供プログラムです。アプリケーション・プログラムでは、DFHJSON にリンクすることによって、JSON をアプリケーション・データに変換できます。

始める前に

JSON バインディングと JSON スキーマを定義する、有効な JSONTRANSFRM リソースが必要です。Java を使用して変換を実行しようとする場合、Axis2 JVM サーバーが既に実行されている必要があります。

このタスクについて

アプリケーション・プログラムを作成または更新して、変換を実行するため、CICS 提供プログラム DFHJSON にリンクします。

手順

1. チャネル (例えば、*MyChannelName*) を作成し、以下のコンテナーをそのチャネルに入れる必要があります。
   - DFHJSON-JSON
   - DFHJSON-TRANSFRM
   - DFHJSON-JVMSERVR

   これらのコンテナーについて詳しくは、JSON 変換プログラム・リンク可能インターフェース・コンテナーを参照してください。
2. **EXEC CICS LINK PROGRAM** API コマンドを使用してデータを JSON に変換します。

   ```
   EXEC CICS LINK PROGRAM('DFHJSON') CHANNEL('MyChannelName')
   ```
3. アプリケーション・プログラムをインストールします。

タスクの結果

アプリケーションが **LINK PROGRAM** コマンドを実行すると、CICS は JSONTRANSFRM リソースをチェックして JSON バインディングのマッピングを検索し、チャネル上のコンテナーを使用して JSON をアプリケーション・データに

変換します。アプリケーション・データは、返されるときに、DFHJSON-DATA ビット・コンテナーに配置されます。

次のタスク

同じマッピングを使用してアプリケーション・データを JSON に変換することもできます。 詳しくは、 528 ページの『DFHJSON へのリンクによるアプリケーション・データの JSON への変換』を参照してください。

TRANSFORM JSONTODATA API コマンドを使用した、JSON のアプリケーション・データへの変換

アプリケーション内で **TRANSFORM JSONTODATA** API コマンドを使用することで、JSON をアプリケーション・データに変換できます。

始める前に

JSON バインディングと JSON スキーマを定義する、有効な JSONTRANSFRM リソースが必要です。Java を使用して変換を実行しようとする場合、Axis2 JVM サーバーが既に実行されている必要があります。

このタスクについて

そのアプリケーションはチャネル・ベースのインターフェースを使用する必要があります。

手順

1. チャネルを作成して、変換対象の JSON が含まれる入力コンテナーをそのチャネルに入れます。

   注: **TRANSFORM JSONTODATA** コマンドが完了すると、変換されたデータが含まれる出力コンテナーもこのチャネルに入れられます。**TRANSFORM JSONTODATA** を発行する前に、出力コンテナーを作成しないでください。このコンテナーは、このコマンドの一環として作成されて、データが取り込まれます。

2. **TRANSFORM JSONTODATA** コマンドを使用して、JSON をアプリケーション・データに変換します。 以下に例を示します。

```
EXEC CICS TRANSFORM JSONTODATA CHANNEL(
ChannelName
) INCONTAINER(
InpContainerName
) OUTCONTAINER(
OutContainerName
) TRANSFORMER(
BundleName
)
```

タスクの結果

アプリケーションが **TRANSFORM JSONTODATA** コマンドを実行すると、CICS は JSONTRANSFRM バンドル・リソースをチェックして JSON バインディング内のマッピングを検索し、チャネル上のコンテナーを使用して JSON をアプリケーション・バイナリー・データに変換します。この変換されたデータが、**TRANSFORM JSONTODATA** コマンドの **OUTCONTAINER** オプションで指定したコンテナーに格納され

て返されます。このオプションを省略すると、デフォルトで DFHJSON-DATA が使用されます。

JSON Web サービス・アプリケーションの作成

JSON を変換するリンク可能インターフェースを使用して RESTful Web サービスを呼び出し、WEB コマンドを使用してその JSON をリモートのサービス・プロバイダーに送信するアプリケーション・プログラムを作成できます。

始める前に

リンク可能インターフェースを使用して JSON を変換する方法をよく理解しておく必要があります。この詳細については、アプリケーション・データおよび JSON のマッピングと変換に記載されています。また、EXEC CICS WEB API についてもよく理解しておく必要があります。この詳細については、HTTP クライアントとしての CICS を介した HTTP 要求に記載されています。

このタスクについて

CICS アプリケーションの一部として、別のシステム上でホストされている RESTful Web サービスを呼び出すことがあります。これを実行するには、リモート・サービスと交換するデータを最初に記述する必要があります。それから、EXEC CICS WEB API を使用して HTTP プロトコルでリモート・サービスと通信する (要求データをサービスに送信し、応答データを受信する) アプリケーション・プログラムを作成できます。リンク可能インターフェースを使用して、要求の一部として使用するためアプリケーション・データを JSON に変換したり、JSON 応答をアプリケーション・データに変換したりできます。一部のサービスでは、要求と応答の両方のペイロードはサポートされない場合があります。

手順

1. リモート・サービスのインターフェースを定義します。
   a. リモート・サービスが存在する場合は、要求および応答のペイロードを記述する JSON スキーマが使用可能であるかどうかを確認します。そうでない場合には、作成する必要があります。 JSON アシスタントを使用して、言語構造へのマッピングを生成します。詳しくは、JSON スキーマからのマッピングの生成を参照してください。
   b. リモート・サービスがまだ存在していない状況で、インターフェースをアプリケーションのデータ構造に基づかせる場合には、JSON アシスタントを使用して JSON スキーマを生成します。その後、JSON スキーマをリモート・サービス・アプリケーション開発者に渡します。詳しくは、言語構造からのマッピングの生成を参照してください。
2. JSON アシスタントによって生成されるバンドルの BUNDLE リソースを定義し、そのバンドルを CICS にインストールします。
3. リモート・サービス・エンドポイントの URIMAP リソースを定義し、インストールします。詳しくは、URIMAP リソースを参照してください。
4. アプリケーション・プログラムを作成または更新して、以下のようにして、リモート・サービスを呼び出します。
   a. リモート・サービスが要求用の JSON ペイロードを必要とする場合 (例えば、HTTP メソッドが POST または PUT の場合)、リンク可能インターフ

ェースを使用してアプリケーション・データを JSON に変換します。 詳しくは、DFHJSON へのリンクによるアプリケーション・データの JSON への変換を参照してください。

b. **EXEC CICS WEB OPEN** コマンドを使用して、リモート・サービスがホストされているサーバーへの接続を開きます。 詳しくは、WEB OPENを参照してください。

c. サービスの要件に応じて、JSON が提供されていることを示すコンテンツ・タイプ・ヘッダー `application/JSON` を指定するように、**EXEC CICS WEB WRITE HTTPHEADER** コマンドをコーディングできます。 詳しくは、WEB WRITE HTTPHEADERを参照してください。

d. 要求をリモート・サービスに送信し、応答を受信するように、**EXEC CICS WEB CONVERSE** コマンドをコーディングします。必要に応じて、照会ストリングまたは要求本体 (DFHJSON-JSON コンテナーから) を指定します。サービスからの応答が予期されている場合、DFHJSON-JSON コンテナーを指定して、応答 JSON を受信します。 詳しくは、WEB CONVERSEを参照してください。

e. その他の要求を行うことを期待しない場合、接続を閉じるように **EXEC CICS WEB CLOSE** コマンドをコーディングします。詳しくは、WEB CLOSE を参照してください。

f. エラーが発生した場合、**EXEC CICS WEB CONVERSE** によって返される HTTP 応答コードをチェックし、適切な処置を行います。 例えば、要求を再試行するか、エラーをユーザーに返します。

g. リモート・サービスからの応答本体が予期されていた場合、リンク可能インターフェースを使用して、JSON をアプリケーション・データに変換します。 詳しくは、DFHJSON へのリンクによるアプリケーション・データの JSON への変換を参照してください。

タスクの結果

SOAP Web サービスの作成

既存の CICS アプリケーションを SOAP Web サービスとして公開し、新規の CICS アプリケーションを作成して SOAP Web サービス・プロバイダーまたはリクエスターとして機能させることができます。

始める前に

SOAP Web サービスの作成を始める前に、以下の作業を行ってください。

1. Web サービスをサポートするよう CICS システムを構成します (Web サービスに応じた CICS システムの構成を参照)。
2. Web サービスの配置をサポートするために必要なインフラストラクチャーを作成します (Web サービス・インフラストラクチャーの作成を参照)。
3. Web サービス・アシスタントを使用するかどうか決定します (SOAP Web サービス使用の計画立案を参照)。

このタスクについて

付属のユーティリティーである CICS Web サービス・アシスタントは、新しい SOAP Web サービス・プロバイダーまたはサービス・リクエスター・アプリケーションに必要な成果物を作成したり、既存のアプリケーションを Web サービス・プロバイダーとして使用可能にする作業を支援します。

CICS Web サービス・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。 この情報は、パイプラインの処理中に、CICS Web サービス・サポートにより使用されます。

以下の手順の記述に従って Web サービスを作成した後、正しく機能することを検証します。

手順

1. 以下の 4 通りのいずれかの方法で SOAP Web サービスを作成します。
   - Web サービス・アシスタントを使用して、Web サービス記述または言語構造を作成して、CICS に配置します。 **PIPELINE SCAN** コマンドを使用して、必要な CICS リソースを自動的に作成します。
   - IBM Developer for Z または Java API を使用して、 Web サービス記述または言語構造を作成して、CICS に配置します。 **PIPELINE SCAN** コマンドを使用して、必要な CICS リソースを自動的に作成します。
   - データ変換を含む、インバウンド・メッセージとアウトバウンド・メッセージの XML を処理し、正しいコンテナーをパイプラインに取り込むためのアプリケーション・プログラムを作成または変更します。 必要な CICS リソースを手動で作成する必要があります。
   - Web サービスとして Axis2 アプリケーションを配置します。
2. Web サービスを開始して、意図したように機能するかをテストします。 Web サービス・アシスタントを使用して Web サービスを配置する場合は、**SET WEBSERVICE** コマンドを使用して検証をオンにすることができます。 この検証によって、データが正しく変換されていることがチェックされます。

次のタスク

これらのステップの詳細については、次のトピックで説明します。

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。

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。

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。

Web サービス・アシスタントのユーティリティー・プログラムとデータ・マッピングの詳細については、以下のトピックを参照してください。

DFHLS2WS: 高水準言語から WSDL への変換

DFHLS2WS プロシージャーは、高水準言語データ構造から 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。

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、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/ に付加されて、/usr/lpp/*path* という完全なパス名が生成されます。

通常、このパラメーターは指定しません。 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。

PATHPREF = *prefix*
他のパラメーターで使用される z/OS UNIX ディレクトリー・パスを拡張するオプションの接頭部を指定します。 デフォルトは空ストリングです。

通常、このパラメーターは指定しません。 デフォルト値は、**PATHPREF** 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。

SERVICE = *value*
このパラメーターは IBM サポートに指示された場合にのみ使用します。

TMPDIR = *tmpdir*
DFHLS2WS が一時ワークスペースとして使用する z/OS UNIX 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 LS2WS です。

USSDIR = *path*
UNIX システム・サービスのファイル・システム内の 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/ に付加されて、/usr/lpp/cicsts/*path* という完全なパス名が生成されます。

通常、このパラメーターは指定しません。 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。

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。

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

/tmp/LS2WS.in

/tmp/LS2WS.out

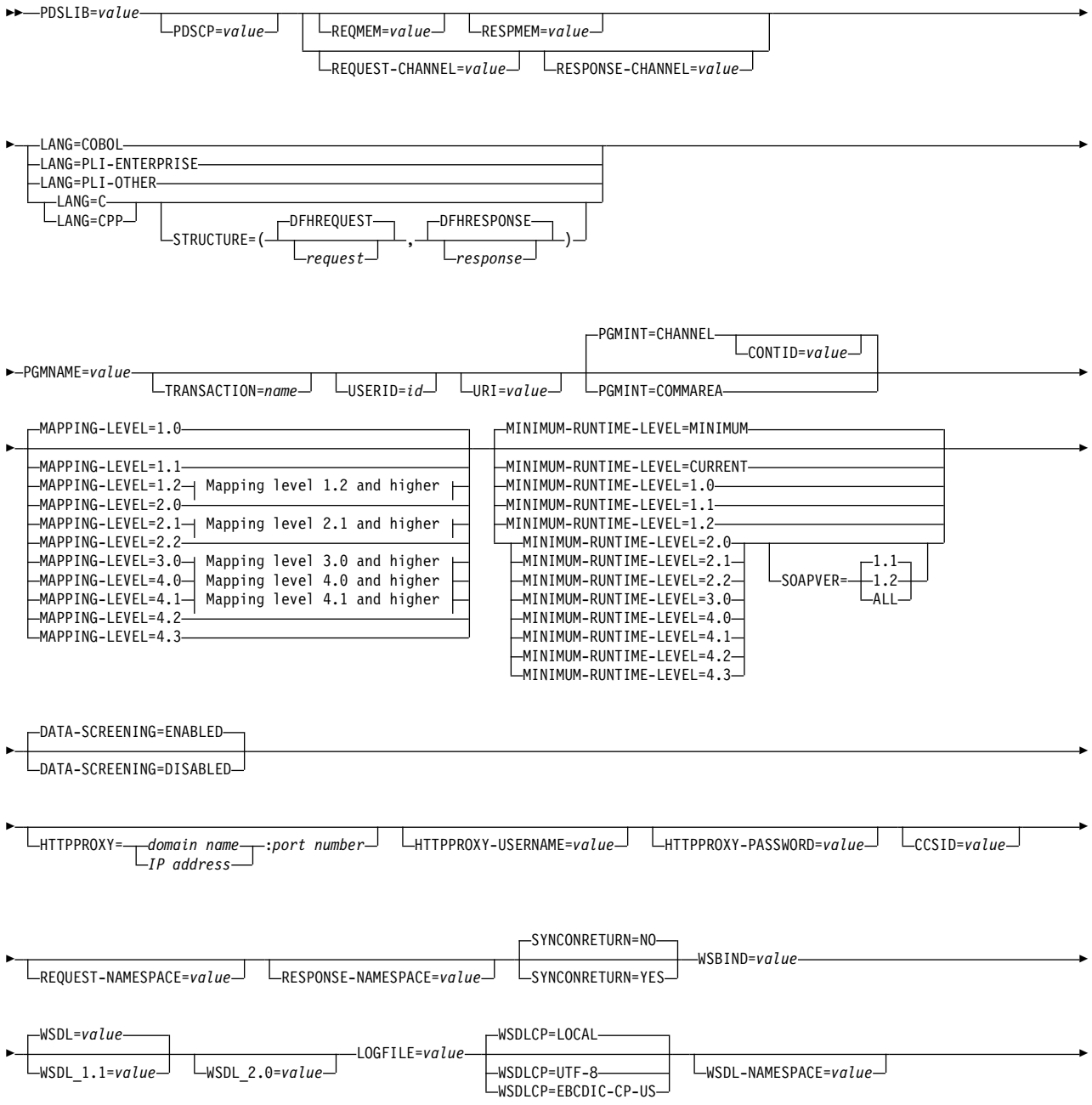
/tmp/LS2WS.err

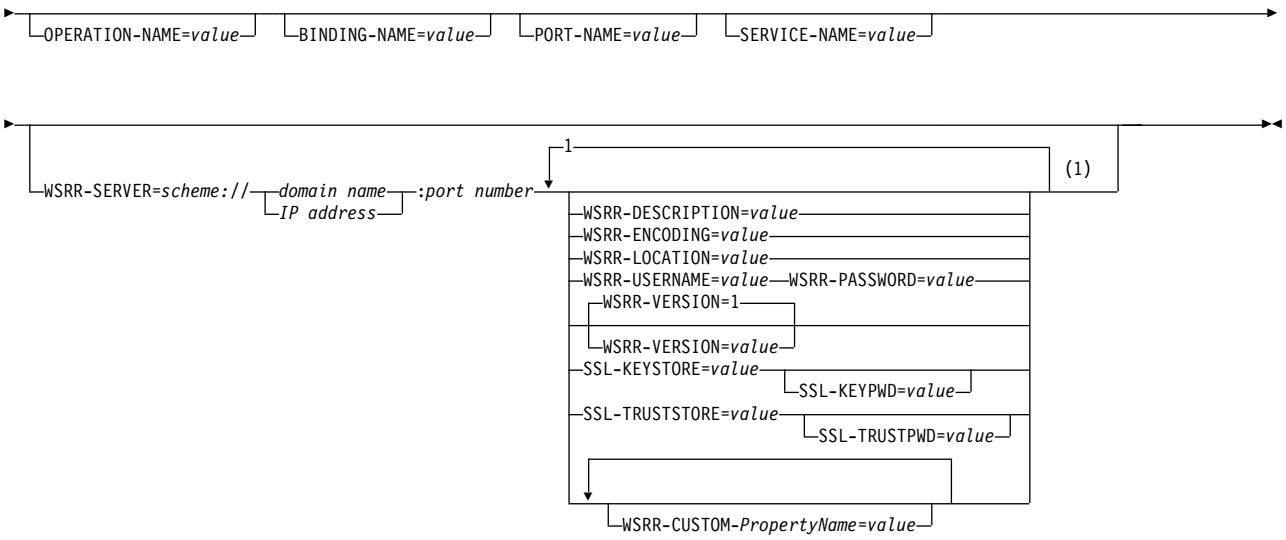
重要: DFHLS2WS は、z/OS UNIX ファイルまたはデータ・セット・メンバーへのアクセスをロックしません。DFHLS2WS の複数のインスタンスが同時に実行され、同じ一時ワークスペース・ファイルを使用する場合、あるジョブがワークスペ

ース・ファイルを使用している間に別のジョブがそれらのファイルを上書きするのを防ぐことはできません。そのため、予測不能な障害が発生する可能性があります。

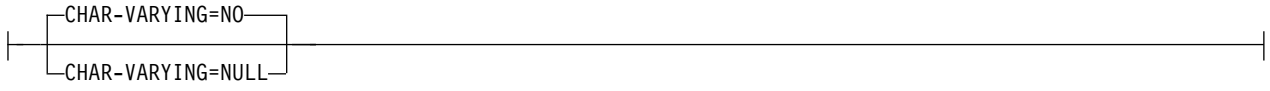
したがって、この状況を回避する命名規則および操作手順を考案することをお勧めします。例えば、システム・シンボリック・パラメーター **SYSUID** を使用すると、個々のユーザーに対して一意のワークスペース・ファイルを生成できます。これらの一時ファイルはジョブの終わりの前に削除されます。

DFHLS2WS のパラメーターの入力



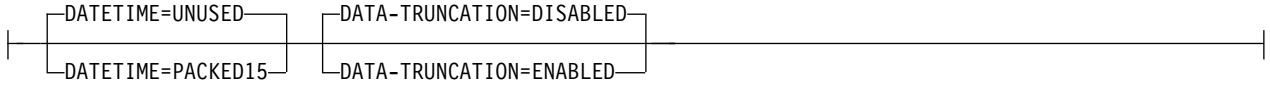


マッピング・レベル **1.2** 以上の場合:

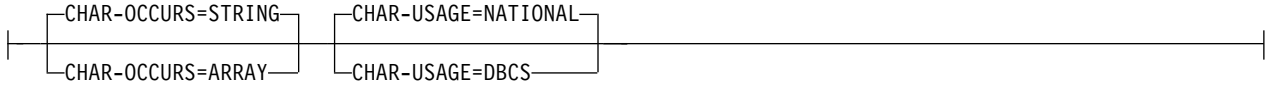


マッピング・レベル **2.1** 以上の場合:

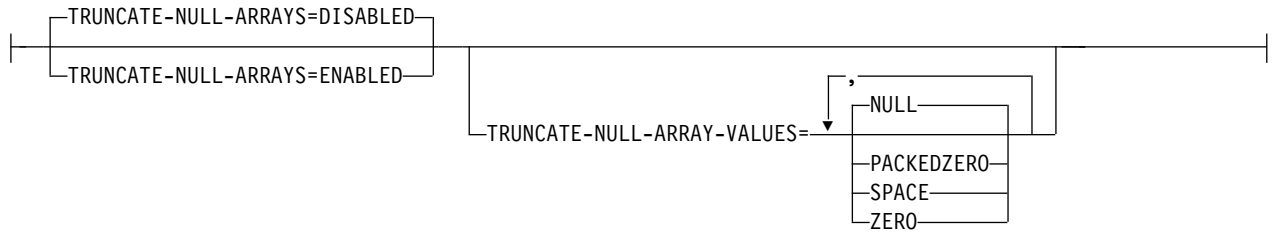
マッピング・レベル **3.0** 以上:



マッピング・レベル **4.0** 以上の場合:



マッピング・レベル **4.1** 以上の場合:



注:

1 **WSRR-SERVER** パラメーターの設定時に指定できるそれぞれの WSRR パラメーターは、一度だけ指定可能です。ただし、例外として、**WSRR-CUSTOM** パラメーターは最大 255 回まで指定可能です。

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。
- パラメーターおよび使用する場合は継続文字は、72 列を超えてはなりません。列 73 から 80 はブランクにする必要があります。
- 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。 スペースを含むアスタリスクより前の文字はすべてパラメーターの一部とみなされます。 以下に例を示します。

```
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。
- 行の末尾の文字の位置にコンマがある場合、これはオプションの行分離文字であり、無視されます。

パラメーターの記述

BINDING-NAME = *value*

生成される WSDL 文書内で使用するバインディング名を指定します。値を指定しない場合、**PGMNAME** パラメーターの値の後に「HTTPSoapBinding」を続けた値を使用してデフォルトのバインディング名が生成されます。SOAPVER が ALL に設定されている場合は、SOAP 1.2 バインディングの名前の後に接尾部「12」が付加されます。

CCSID = *value*

アプリケーション・データ構造に文字データをエンコードするために、実行時に使用される CCSID を指定します。このパラメーターの値は、**LOCALCCSID** システム初期設定パラメーターの値を指定変更します。 *value* は、Java および z/OS 変換サービスによってサポートされている EBCDIC CCSID である必要があります (z/OS Unicode Services ユーザーズ・ガイドおよび解説書を参

照)。このパラメーターを指定しない場合、アプリケーション・データ構造は、システム初期設定パラメーターで指定された CCSID を使用してエンコードされます。

このパラメーターは任意のマッピング・レベルで使用できます。

CHAR-VARYING = { **NO** | **NULL** | **COLLAPSE** | **BINARY** }

マッピング・レベルが 1.2 以上のとき、言語構造内の文字フィールドがマップされる方法を指定します。 COBOL の文字フィールドは、タイプ X のピクチャー節 (例えば `PIC(X) 10`) です。C/C++ の文字フィールドは文字配列です。以下のオプションを選択できます。

NO 文字フィールドは <xsd:string> にマップされ、固定長のフィールドとして処理されます。 データの最大長はフィールドの長さと同じです。 NO は、マッピング・レベル 2.0 以前での、COBOL および PL/I の **CHAR-VARYING** パラメーターのデフォルト値です。

この値は、Enterprise およびその他の PL/I 言語構造に適用されません。

NULL

文字フィールドは <xsd:string> にマップされ、ヌル終了ストリングとして処理されます。 CICS は、SOAP メッセージから変換する際に、終了ヌル文字を追加します。 文字ストリングの最大長は、言語構造に示される長さより 1 文字少ない長さとして計算されます。 NULL は、C/C++ での **CHAR-VARYING** パラメーターのデフォルト値です。

この値は、Enterprise およびその他の PL/I 言語構造に適用されません。

COLLAPSE

文字フィールドは <xsd:string> にマップされます。フィールド内の後続空白と埋め込み空白は SOAP メッセージに含まれません。例えば、`<space>AB<space><space><space>C<space>` は `AB<space>C` になります。インバウンド SOAP メッセージは解析され、先行空白、後続空白、埋め込み空白はすべて削除されます。マッピング・レベル 2.1 以降の COBOL および PL/I では、**CHAR-VARYING** パラメーターのデフォルト値は COLLAPSE です。

可変長の値と空白についての詳細は、可変長の値と空白のサポートを参照してください。

BINARY

文字フィールドは <xsd:base64binary> にマップされ、固定長のフィールドとして処理されます。 **CHAR-VARYING** パラメーターの BINARY 値は、マッピング・レベル 2.1 以降でのみ使用可能です。

CHAR-OCCURS = { **STRING** | **ARRAY** }

マッピング・レベル 4.0 以上の場合に、言語構造内の文字配列をマップする方法を指定します。例えば `PIC X OCCURS 20` となります。このパラメーターは COBOL 言語でのみ使用されます。

ARRAY
文字配列は XML 配列にマップされます。つまり、それぞれの文字が個別の XML エレメントとしてマップされます。マッピング・レベル 3.0 以前でも、このように動作します。

STRING
文字配列は XML ストリングにマップされます。つまり、COBOL 配列全体が 1 つの XML エレメントとしてマップされます。

CHAR-USAGE = { **NATIONAL** | **DBCS** }

COBOL では、各国語データ型 PIC N を UTF-16 または DBCS データ用に使用できます。この設定は NSYMBOL コンパイラー・オプションによって制御されます。データが適切に扱われるようにするには、アシスタントの **CHAR-USAGE** パラメーターを NSYMBOL コンパイラー・オプションと同じ値に設定する必要があります。 UTF-16 を使用する場合には、通常、これは CHAR-USAGE=NATIONAL に設定されます。

DBCS
PIC (*n*) フィールドからのデータは、DBCS でエンコードされたデータとして扱われます。

NATIONAL
PIC (*n*) フィールドからのデータは、UTF-16 でエンコードされたデータとして扱われます。

CONTID = *value*
サービス・プロバイダーで、SOAP メッセージを表示するために使用される最上位のデータ構造を格納するコンテナーの名前を指定します。

CICS がターゲット・アプリケーション・プログラムに渡すコンテナーの長さは、要求コンテナーの長さおよび応答コンテナーの長さより大きくなります。

DATA-SCREENING = { **ENABLED** | **DISABLED** }
アプリケーション提供のデータのエラーを検査するかどうかを指定します。

ENABLED
言語構造と矛盾するアプリケーション提供のランタイム・データは、エラーとして扱われ、メッセージ DFHPI1010 が発行されます。エラー応答がアプリケーションに 返されます。

DISABLED
言語構造と矛盾するアプリケーション提供のランタイム・データの値は、 デフォルト値で置き換えられます。例えば、数値フィールド内のゼロは誤った値を置き換えます。メッセージ DFHPI1010 は発行されず、通常の応答がアプリケーションに返されます。この機能を使用して、 初期設定されていない出力フィールドから生成される INVALID_PACKED_DEC および INVALID_ZONED_DEC エラー応答を回避できます。

DATA-TRUNCATION = { **DISABLED** | **ENABLED** }
固定長フィールド構造で可変長データを許容するかどうかを指定します。

DISABLED
データが CICS が予期する固定長より短い場合に、CICS は切り捨てられたデータを拒否してエラー・メッセージを発行します。

ENABLED
データが CICS が予期する固定長より短い場合に、CICS は切り捨てられたデータを許容して、欠落データをヌル値として処理します。

DATETIME = { **UNUSED** | **PACKED15** }
高水準言語構造内で ABSTIME になる可能性のあるフィールドをタイム・スタンプとしてマップするかどうかを指定します。

PACKED15
長さ 15 (8 バイト) のパック 10 進数フィールドは CICS の ABSTIME フィールドとして扱われ、タイム・スタンプとしてマップされます。

UNUSED
長さ 15 (8 バイト) のパック 10 進数フィールドは、タイム・スタンプとしては扱われません。

このパラメーターは、マッピング・レベル 3.0 で設定できます。

HTTPPROXY = { *domain name* **:** *port number* | *IP 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、DFHLS2WS 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。 以下に例を示します。

```
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。

HTTPPROXY-PASSWORD = *value*
HTTP プロキシー・パスワードを指定します。 DFHLS2WS を実行するシステムが HTTP プロキシー・サーバーを使ってインターネットにアクセスする場合、HTTP プロキシー・サーバーが基本認証を使用するならば、**HTTPPROXY-USERNAME** と共にこのパスワードを使用する必要があります。 **HTTPPROXY** も指定した場合にのみ、このパラメーターを使用できます。

HTTPPROXY-USERNAME = *value*
HTTP プロキシー・ユーザー名を指定します。 DFHLS2WS を実行するシステムが HTTP プロキシー・サーバーを使ってインターネットにアクセスする場合、HTTP プロキシー・サーバーが基本認証を使用するならば、**HTTPPROXY-PASSWORD** と共にこのユーザー名を使用する必要があります。 **HTTPPROXY** も指定した場合にのみ、このパラメーターを使用できます。

LANG = **COBOL**|**PLI-ENTERPRISE**|**PLI-OTHER**|**C**|**CPP**
高水準言語構造のプログラミング言語を指定します。

COBOL
COBOL

PLI-ENTERPRISE
Enterprise PL/I

PLI-OTHER
Enterprise PL/I 以外の PL/I のレベル

C C

CPP C++

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 z/OS UNIX 名です。DFHLS2WS は、このファイルが存在しない場合、ディレクトリー構造は作成しませんがファイルを作成します。

通常、このファイルは使用しませんが、DFHLS2WS で問題が発生した場合に IBM サービス組織から要求されることがあります。

MAPPING-LEVEL = { **1.0** | **1.1** | **1.2** | **2.0** | **2.1** | **2.2** | **3.0** | **4.0** | **4.1** | **4.2** | **4.3** }

Web サービス・バインディング・ファイルおよび Web サービス記述を生成するときに、DFHLS2WS が使用するマッピングのレベルを指定します。 以下のオプションを選択できます。

1.0 このマッピング・レベルはデフォルトです。 これは Web サービス・バインディング・ファイルが CICS TS 3.1 マッピング・レベルを使用して生成されることを示します。

1.1 このマッピングは、この特定のレベルでバインディング・ファイルを再生成するために使用します。

1.2 このマッピング・レベルでは、**CHAR-VARYING** パラメーターを使用して、実行時に文字配列が処理される方法を制御します。VARYING と VARYINGZ 配列は PL/I でもサポートされます。

2.0 このマッピング・レベルは、CICS TS 3.2 領域またはそれ以降で、言語構造と Web サービス・バインディング・ファイル間のマッピングに対する機能拡張を利用するときに使用します。

2.1 このマッピング・レベルは、CICS TS 3.2 以降の領域で使用します。このマッピング・レベルでは、**CHAR-VARYING** パラメーターの新しい値 COLLAPSE および BINARY を利用できます。COBOL の FILLER フィールドおよび PL/I の * フィールドは、このマッピング・レベルでは体系的に無視され、それらのフィールドは生成された WSDL には表示されず、データ構造には実行時に適切なギャップが残されます。

2.2 このマッピング・レベルは、CICS TS 3.2 以降の領域で、DFHWS2LS 使用時のマッピングの機能拡張を利用するときに使用します。

3.0 このマッピング・レベルは、CICS TS 4.1 以降の領域で使用します。このマッピング・レベルでは、**REQUEST-CHANNEL** パラメーターと **RESPONSE-CHANNEL** パラメーターを設定することにより、インターフェースで多数のコンテナーを使用するアプリケーションから Web サービスを作成することができます。 **DATETIME** パラメーターを設定することで、dateTime フィールドを XML タイム・スタンプにマップすることもできます。

4.0 このマッピング・レベルは、CICS TS 5.2 以降の領域で使用します。このマッピング・レベルでは、COBOL の OCCURS DEPENDING ON フィールドおよび **CHAR-OCCURS** パラメーターを使用できます。

4.1 切り捨て可能な配列をサポートするには、CICS TS 5.2 以降の領域でこのマッピング・レベルを使用します。

4.2 大きな変更はありません。このマッピング・レベルは、CICS TS 5.4 以降の領域で使用します。

4.3 大きな変更はありません。このマッピング・レベルは、CICS TS 5.4 以降の領域で使用します。

マッピング・レベルについて詳しくは、CICS アシスタントのマッピング・レベルを参照してください。

MINIMUM-RUNTIME-LEVEL = { MINIMUM | 1.0 | 1.1 | 1.2 | 2.0 | 2.1 | 2.2 | 3.0 | 4.0 | 4.1 | 4.2 | 4.3 | CURRENT }

Web サービス・バインディング・ファイルを配置できる最小の CICS 実行時環境を指定します。指定した他のパラメーターと一致しないレベルを選択すると、エラー・メッセージを受け取ります。以下のオプションを選択できます。

MINIMUM
選択したパラメーターを前提として、最低限可能な CICS 実行時レベルが自動的に割り振られます。

1.0 生成された Web サービス・バインディング・ファイルが、CICS TS 3.1 領域で正常に配置されます。一部のパラメーターは、このランタイム・レベルでは使用できません。

1.1 生成された Web サービス・バインディング・ファイルが、CICS TS 3.1 領域で正常に配置されます。**MAPPING-LEVEL** パラメーターには、マッピング・レベル 1.1 以前を使用できます。一部のパラメーターは、このランタイム・レベルでは使用できません。

1.2 生成された Web サービス・バインディング・ファイルが、CICS TS 3.1 領域で正常に配置されます。**MAPPING-LEVEL** パラメーターには、マッピング・レベル 1.2 以前を使用できます。一部のパラメーターは、このランタイム・レベルでは使用できません。

2.0 生成された Web サービス・バインディング・ファイルが、CICS TS 3.2 以降の領域で正常に配置されます。 **MAPPING-LEVEL** パラメーターには、マッピング・レベル 2.0 以前を使用できます。一部のパラメーターは、このランタイム・レベルでは使用できません。

2.1 生成された Web サービス・バインディング・ファイルは、CICS TS 3.2 以降の領域に正常にデプロイされます。**MAPPING-LEVEL** パラメーターには、マッピング・レベル 2.1 以前を使用できます。一部のパラメーターは、このランタイム・レベルでは使用できません。

2.2 生成された Web サービス・バインディング・ファイルは、CICS TS 3.2 以降の領域に正常にデプロイされます。この実行時レベルでは、**MAPPING-LEVEL** パラメーターには、マッピング・レベル 2.2 以前を使用することができます。一部のパラメーターは、このランタイム・レベルでは使用できません。

3.0 生成された Web サービス・バインディング・ファイルは、CICS TS 4.1 以降の領域に正常にデプロイされます。この実行時レベルでは、**MAPPING-LEVEL** パラメーターには、マッピング・レベル 3.0 以前を使用することができます。一部のパラメーターは、このランタイム・レベルでは使用できません。

4.0 生成された Web サービス・バインディング・ファイルは、CICS TS 5.2 以降の領域に正常にデプロイされます。この実行時レベルでは、

MAPPING-LEVEL パラメーターにマッピング・レベル 4.0 以前を使用できます。このレベルでは任意のオプション・パラメーターを使用できます。

4.1 生成された Web サービス・バインディング・ファイルは、CICS TS 5.2 以降の領域に正常にデプロイされます。この実行時レベルでは、**MAPPING-LEVEL** パラメーターには、マッピング・レベル 4.1 以前を使用することができます。

4.2 生成された Web サービス・バインディング・ファイルは、CICS TS V5.4 以降の領域に正常にデプロイされます。この実行時レベルでは、**MAPPING-LEVEL** パラメーターには、マッピング・レベル 4.2 以前を使用することができます。

4.3 生成された Web サービス・バインディング・ファイルは、CICS TS 5.4 以降の領域に正常にデプロイされます。この実行時レベルでは、**MAPPING-LEVEL** パラメーターには、マッピング・レベル 4.3 以前を使用することができます。

CURRENT
生成された Web サービス・バインディング・ファイルは、Web サービス・バインディング・ファイルの生成に使用するものと同じ実行時レベルで、CICS 領域に正常に配置されます。

OPERATION-NAME = *value*
生成される WSDL 文書で使用される操作名を指定します。 値が提供されていない場合には、**PGMNAME** パラメーターの値に続いて **operation** 値を使用して、デフォルト名が生成されます。

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。

PDSCP = *value*
REQMEM および **RESPMEM** パラメーターに指定される区分データ・セット・メンバーで使用されるコード・ページを指定します。ここで、*value* は CCSID 番号または Java コード・ページ番号です。このパラメーターを指定しない場合は、z/OS UNIX システム・サービスのコード・ページが使用されます。 例えば、**PDSCP**=037 と指定できます。

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ターゲット・アプリケーション・プログラムにデータを渡します。

- マッピング・レベル 3.0 より前では、チャネルに含められるのは 1 つのコンテナーのみで、このコンテナーは入出力の両方に使用されま

す。 **CONTID** パラメーターを使用してコンテナーの名前を指定します。 デフォルト名は DFHWS-DATA です。

- マッピング・レベル 3.0 では、チャネルに複数のコンテナーを含めることができます。 **REQUEST-CHANNEL** および **RESPONSE-CHANNEL** パラメーターを使用します。 **PDSLIB**、**REQMEM**、および **RESPMEM** は指定しないでください。

COMMAREA

CICS は、通信域 (COMMAREA) 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。

ターゲット・アプリケーション・プログラムが要求を処理するとき、同じメカニズムを使用して応答を戻す必要があります。要求が通信域で受信されている場合は、応答を通信域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、応答をコンテナーに戻す必要があります。CICS がターゲット・アプリケーション・プログラムに渡す通信域またはコンテナーの長さは、要求の通信域またはコンテナーの長さ、および応答の通信域またはコンテナーの長さより大きくなります。

PGMNAME = *value*

Web サービスとして公開されるターゲット・アプリケーション・プログラム用の CICS PROGRAM リソースの名前を指定します。CICS Web サービスのサポートがこのプログラムにリンクします。

PORT-NAME = *value*

生成される WSDL 文書内で port および portType に使用する名前を指定します。値を指定しない場合、**PGMNAME** パラメーターの値の後に「Port」を続けた値を使用してデフォルトの名前が生成されます。SOAPVER が ALL に設定されている場合は、SOAP 1.2 ポートの名前の後に接尾部「12」が付加されます。

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。

REQUEST-CHANNEL = *value*

チャネル記述文書の名前と場所を指定します。チャネル記述では、Web サービス・リクエスターから SOAP メッセージを受信する際に Web サービス・プロバイダー・アプリケーションがそのインターフェースで使用できるコンテナーについて記述します。 チャネル記述は、CICS 提供のチャネル・スキーマに準拠する必要のある XML 文書です。

このパラメーターはマッピング・レベル 3.0 のみで使用できます。

REQUEST-NAMESPACE = *value*

生成される Web サービス記述に含まれている要求メッセージの XML スキーマのネーム・スペースを指定します。 このパラメーターを指定しない場合、CICS では名前空間が自動的に生成されます。

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。

応答が存在しない場合 (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。

RESPONSE-CHANNEL = *value*
チャネル記述文書の名前と場所を指定します。チャネル記述では、Web サービス・リクエスターに SOAP 応答メッセージを送信する際に Web サービス・プロバイダー・アプリケーションがそのインターフェースで使用できるコンテナーについて記述します。 チャネル記述は、CICS 提供のチャネル・スキーマに準拠する必要のある XML 文書です。

このパラメーターはマッピング・レベル 3.0 のみで使用できます。

RESPONSE-NAMESPACE = *value*
生成される Web サービス記述に含まれている応答メッセージの XML スキーマのネーム・スペースを指定します。 このパラメーターを指定しない場合、CICS では名前空間が自動的に生成されます。

SERVICE-NAME = *value*
生成される WSDL 文書内で使用するサービス名を指定します。値を指定しない場合、**PGMNAME** パラメーターの値の後に「Service」を続けた値を使用してデフォルトのサービス名が生成されます。

SOAPVER = { **1.1** | **1.2** | **ALL** }
生成される Web サービス記述で使用する SOAP レベルを指定します。このパラメーターは、**MINIMUM-RUNTIME-LEVEL** が 2.0 以上に設定されている場合にのみ使用可能です。

1.1 Web サービス記述のバインディングとして SOAP 1.1 プロトコルを使用します。

1.2 Web サービス記述のバインディングとして SOAP 1.2 プロトコルを使用します。

ALL Web サービス記述のバインディングとして SOAP 1.1 プロトコルと 1.2 プロトコルの両方を使用できます。

このパラメーターに値を指定しない場合は、作成したい WSDL のバージョンに応じてデフォルト値が異なります。

- WSDL 1.1 だけが必要な場合は、SOAP 1.1 バインディングが使用されます。
- WSDL 2.0 だけが必要な場合は、SOAP 1.2 バインディングが使用されます。
- WSDL 1.1 と WSDL 2.0 の両方が必要な場合は、それぞれの Web サービス記述で SOAP 1.1 と 1.2 の両方のバインディングが使用されます。

SSL-KEYSTORE = *value*
このオプション・パラメーターは、鍵ストア・ファイルの完全修飾された場所を指定します。

Web サービス・アシスタントが SSL (Secure Sockets Layer) 暗号化を使用して、ネットワークを介して IBM WebSphere Service Registry and Repository (WSRR) と通信できるようにする場合、このパラメーターを使用します。

SSL-KEYPWD = *value*
このオプション・パラメーターは、鍵ストアのパスワードを指定します。

Web サービス・アシスタントが SSL (Secure Sockets Layer) 暗号化を使用して、ネットワークを介して IBM WebSphere Service Registry and Repository (WSRR) と通信できるようにする場合、このパラメーターを使用します。

SSL-TRUSTSTORE = *value*
このオプション・パラメーターは、トラストストア・ファイルの完全修飾された場所を指定します。

Web サービス・アシスタントが SSL (Secure Sockets Layer) 暗号化を使用して、ネットワークを介して IBM WebSphere Service Registry and Repository (WSRR) と通信できるようにする場合、このパラメーターを使用します。

SSL-TRUSTPWD = *value*
このオプション・パラメーターは、トラストストアのパスワードを指定します。

Web サービス・アシスタントが SSL (Secure Sockets Layer) 暗号化を使用して、ネットワークを介して IBM WebSphere Service Registry and Repository (WSRR) と通信できるようにする場合、このパラメーターを使用します。

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や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。

応答
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や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。

SYNCONRETURN = { **NO** | **YES** }
リモート Web サービスが同期点を発行できるかどうかを指定します。

NO リモート Web サービスは同期点を発行できません。 この値はデフォルトです。リモート Web サービスが同期点を発行すると、ADPL 異常終了になり失敗します。

YES リモート Web サービスは同期点を発行できます。 YES を選択した場合、リモート Web サービスから制御が戻されたときにリモート・タスクが別個の作業単位としてコミットされます。リモート Web サービスがリカバリー可能リソースを更新して戻した後に障害が発生した場合、そのリソースの更新内容をバックアウトすることはできません。

TRANSACTION = *name*
サービス・プロバイダーで、このパラメーターは、応答を組み立てるためにパイプラインを開始できる 1 から 4 文字の別名トランザクションの名前を指定しま

す。 このパラメーターの値は、URIMAP リソースが **PIPELINE** スキャン・コマンドを使用して自動的に作成される際に、URIMAP リソースの TRANSACTION 属性を定義するために使用されます。

許容文字: A-Z a-z 0-9 $ @ # _ < >

TRUNCATE-NULL-ARRAYS = { **DISABLED** | **ENABLED** }

マッピング・レベル 4.1 以上で構造化配列を処理する方法を指定します。この機能を有効にすると、CICS は配列に含まれる空のレコードを認識しようと試みます (空のレコードの認識について詳しくは、TRUNCATE-NULL-ARRAY-VALUES を参照)。空の配列レコードが 5 個連続して検出された場合、その配列は XML/JSON の生成時に最初の空のレコードの箇所で切り捨てられます。この機能は、内容が構造化されている配列に対してのみ有効にされます。単純なプリミティブ・フィールドからなる配列には、切り捨ては適用されません。配列の切り捨てにより、JSON/XML 内のデータ表現が簡潔になりますが、リスクがないわけではありません。5 個の連続したデータ・レコードが、(おそらく、正当な低値を含んでいるために) 初期化されていないストレージとして誤って認識された場合、データ損失が生じるおそれがあります。TRUNCATE-NULL-ARRAYS が有効にされていて、TRUNCATE-NULL-ARRAY-VALUES が設定されていない場合は、TRUNCATE-NULL-ARRAY-VALUES のデフォルト値が使用されます。

TRUNCATE-NULL-ARRAY-VALUES = { **NULL** | **PACKEDZERO**|**SPACE** | **ZERO** }

マッピング・レベル 4.1 以上で、TRUNCATE-NULL-ARRAYS の処理で空として扱う値を指定します。デフォルトでは、ヌル値 (0x00、または小さい値) が空として扱われます。構造化された配列のレコードに含まれるすべてのストレージ・バイトにヌルが含まれている場合、レコード全体が空であると見なされます。1 つ以上の NULL、PACKEDZERO、SPACE、および ZERO 値をコンマ区切りリストに指定できます。

NULL
ヌル文字 (0x00) を暗黙指定します。

PACKEDZERO
正符号付きパック 10 進数ゼロ (0x0C)、負符号付きパック 10 進数ゼロ (0x0D)、または符号なしパック 10 進数ゼロ (0x0F) を暗黙指定します。

SPACE
SBCS EBCDIC スペース (0x40) を暗黙指定します。

ZERO
符号なしのゾーン 10 進数ゼロ (0xF0) を暗黙指定します。

構造化配列レコードに、選択したバイトの組み合わせとの一致が含まれている場合、レコード全体が空として識別されます。

TRUNCATE-NULL-ARRAY-VALUES に値が定義されている場合、
TRUNCATE-NULL-ARRAYS が有効に設定されている必要があります。

URI = *value*

このパラメーターは、クライアントが Web サービスにアクセスするときに使

用する相対または絶対 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際に、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 のパスのコンポーネントを指定します。

USERID = *id*
サービス・プロバイダーでは、このパラメーターは、任意の Web クライアントで使用可能な 1 文字から 8 文字のユーザー ID を指定します。 アプリケーションから生成される応答や Web サービスについては、そのユーザー ID の下で別名トランザクションが追加されます。このパラメーターの値は、URIMAP リソースが **PIPELINE** スキャン・コマンドを使用して自動的に作成される際に、URIMAP リソースの USERID 属性を定義するために使用されます。

許容文字: A-Z a-z 0-9 $ @ #

WSBIND = *value*
Web サービス・バインディング・ファイルの完全修飾 z/OS UNIX 名です。DFHLS2WS は、このファイルが存在しない場合、ディレクトリー構造は作成しませんがファイルを作成します。 ファイル拡張子は .wsbind です。

WSDL = *value*
Web サービス記述を書き込むファイルの完全修飾 z/OS UNIX 名です。Web サービス記述は WSDL 1.1 仕様に準拠します。 DFHLS2WS は、このファイルが存在しない場合、ディレクトリー構造は作成しませんがファイルを作成します。 ファイル拡張子は .wsdl です。

WSDL_1.1 = *value*
Web サービス記述を書き込むファイルの完全修飾 z/OS UNIX 名です。Web サービス記述は WSDL 1.1 仕様に準拠します。 DFHLS2WS は、このファイルが存在しない場合、ディレクトリー構造は作成しませんがファイルを作成します。 ファイル拡張子は .wsdl です。 このパラメーターと **WSDL** パラメーターの結果は同じです。どちらか 1 つのみを指定できます。

WSDL_2.0 = *value*
Web サービス記述を書き込むファイルの完全修飾 z/OS UNIX 名です。Web サービス記述は WSDL 2.0 仕様に準拠します。 DFHLS2WS は、このファイルが存在しない場合、ディレクトリー構造は作成しませんがファイルを作成します。 ファイル拡張子は .wsdl です。 このパラメーターは、**WSDL** または **WSDL_1.1** パラメーターとともに使用できます。このパラメーターは **MINIMUM-RUNTIME-LEVEL** が 2.0 以上に設定されている場合にのみ使用可能です。

WSDLCP = { **LOCAL** | **UTF-8** | **EBCDIC-CP-US** }
WSDL 文書を生成するために使用するコード・ページを指定します。

LOCAL
WSDL 文書をローカル・コード・ページを使用して生成し、WSDL 文書にはエンコード・タグが生成されないように指定します。

UTF-8
WSDL 文書を UTF-8 コード・ページを使用して生成するように指定し

ます。 エンコード・タグが WSDL 文書に生成されます。 このオプションを指定する場合、異なるプラットフォーム間で WSDL 文書をコピーするときに正しいエンコードが保持されることを確認する必要があります。

EBCDIC-CP-US
この値は、WSDL 文書を US EBCDIC コード・ページを使用して生成するように指定します。 エンコード・タグが WSDL 文書に生成されます。

WSDL-NAMESPACE = *value*
生成される WSDL 文書で使われる CICS の名前空間を指定します。

このパラメーターを指定しない場合、CICS では名前空間が自動的に生成されます。

WSRR-CUSTOM- *PropertyName* = *value*
このオプション・パラメーターを使用して、カスタマイズされたメタデータを WSRR の WSDL 文書に追加できます。 WSDL 文書に WSRR-CUSTOM-*PropertyName=value* という組が追加されて、WSSR-CUSTOM 接頭部なしで WSRR に表示されます。

最大で 255 個のカスタマイズされた *PropertyName=value* という組を指定できます。重複する *PropertyName=value* の組、およびブランクの組を指定しないでください。

このパラメーターは、**WSRR-SERVER** パラメーターが指定された場合にのみ使用します。

WSRR-DESCRIPTION = *value*
このオプション・パラメーターを使用すると、公開される WSDL 文書を記述するメタデータを指定できます。

このパラメーターは、**WSRR-SERVER** パラメーターが指定された場合にのみ使用します。

WSRR-ENCODING = *value*
このオプション・パラメーターを使用して、WSDL 文書の文字セット・エンコードを指定します。 **WSRR-ENCODING** パラメーターが指定されない場合、WSRR は WSDL 文書で指定された値を使用します。

このパラメーターは、**WSRR-SERVER** パラメーターが指定された場合にのみ使用します。

WSRR-LOCATION = *value*
このオプション・パラメーターを使用して、WSDL 文書の場所を識別する URI を指定します。 このパラメーターを指定しない場合、URI は **WSDL** パラメーターで指定したデフォルトのファイル名になります。例えば **WSDL** パラメーターの値が `wsrr/example.wsdl` である場合、**WSRR-LOCATION** パラメーターのデフォルト値は `example.wsdl` になります。

このパラメーターは、**WSRR-SERVER** パラメーターが指定された場合にのみ使用します。

WSRR-PASSWORD = *value*
WSRR へのアクセスにパスワードの入力が必要な場合は、このオプション・パラメーターを使用します。

WSRR-USERNAME パラメーターを指定する場合には、このパラメーターも指定する必要があります。

このパラメーターは、**WSRR-SERVER** パラメーターが指定された場合にのみ使用します。

WSRR-SERVER = { *domain name* **:** *port number* | *IP address* **:** *port number* }
このパラメーターを使用して、IBM WebSphere Service Registry and Repository (WSRR) サーバーの場所を指定します。このパラメーターを指定した場合、WSRR パラメーター検証が使用されます。

WSRR-USERNAME = *value*
WSRR へのアクセスでユーザー名を指定する必要がある場合は、このオプション・パラメーターを使用します。 WSRR はこのユーザー名を使って所有者プロパティーを設定します。

このパラメーターは、**WSRR-SERVER** パラメーターが指定された場合にのみ使用します。

WSRR-VERSION = { **1** | *value* }
このパラメーターを使用して、WSRR の WSDL 文書のバージョン・プロパティーを設定します。

このパラメーターは、**WSRR-SERVER** パラメーターが指定された場合にのみ使用します。

その他の情報

- DFHLS2SC を実行するユーザー ID は、UNIX システム・サービスを使用するように構成されていなければなりません。 ユーザー ID には、CICS z/OS UNIX ファイル構造および PDS ライブラリーに対する読み取り権限、および **LOGFILE**、**WSBIND**、および **WSDL** パラメーターで指定されたディレクトリーへの書き込み権限が必要です。
- Java を実行するのに十分な大きさのストレージがユーザー ID に割り振られている必要があります。
- JCL の最大パラメーター長は 100 文字です。 これは、**STDPARM** ステートメントを使用して増やすことができます。詳しくは、「z/OS UNIX システム・サービス ユーザーズ・ガイド」を参照してください。

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
LOGFILE=/u/exampleapp/wsbind/example.log
```

```
MINIMUM-RUNTIME-LEVEL=2.1
MAPPING-LEVEL=2.1
CHAR-VARYING=COLLAPSE
PGMNAME=DFH0XCMN
URI=http://myserver.example.org:8080/exampleApp/example
PGMINT=COMMAREA
SOAPVER=1.1
SYNCONRETURN=YES
WSBIND=/u/exampleapp/wsbind/example.wsbind
WSDL=/u/exampleapp/wsdl/example.wsdl
WSDL_2.0=/u/exampleapp/wsdl/example_20.wsdl
WSDLCP=LOCAL
WSDL-NAMESPACE=http://mywsdlnamespace
/*
```

DFHWS2LS: WSDL から高水準言語への変換

DFHWS2LS プロシージャーは Web サービス記述から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。

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、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/ に付加されて、/usr/lpp/*path* という完全なパス名が生成されます。

通常、このパラメーターは指定しません。 デフォルト値は、**JAVADIR** パラメーターで CICS インストール・ジョブ (DFHISTAR) に指定された値です。

PATHPREF = *prefix*
他のパラメーターで使用される z/OS UNIX ディレクトリー・パスを拡張するオプションの接頭部を指定します。 デフォルトは空ストリングです。

通常、このパラメーターは指定しません。 デフォルト値は、**PATHPREF** 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 z/OS UNIX 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 WS2LS です。

USSDIR = *path*
UNIX システム・サービスのファイル・システム内の 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/ に付加されて、/usr/lpp/cicsts/*path* という完全なパス名が生成されます。

通常、このパラメーターは指定しません。 デフォルト値は、**USSDIR** パラメーターで CICS インストール・ジョブ (DFHISTAR) に指定された値です。

SERVICE = *value*
このパラメーターは IBM サポートに指示された場合にのみ使用します。

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

各部の意味は次のとおりです。

tmpdir **TMPDIR** パラメーターで指定された値です。

tmpprefix **TMPFILE** パラメーターで指定された値です。

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

/tmp/WS2LS.in

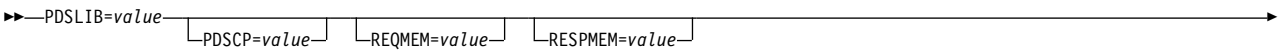
/tmp/WS2LS.out

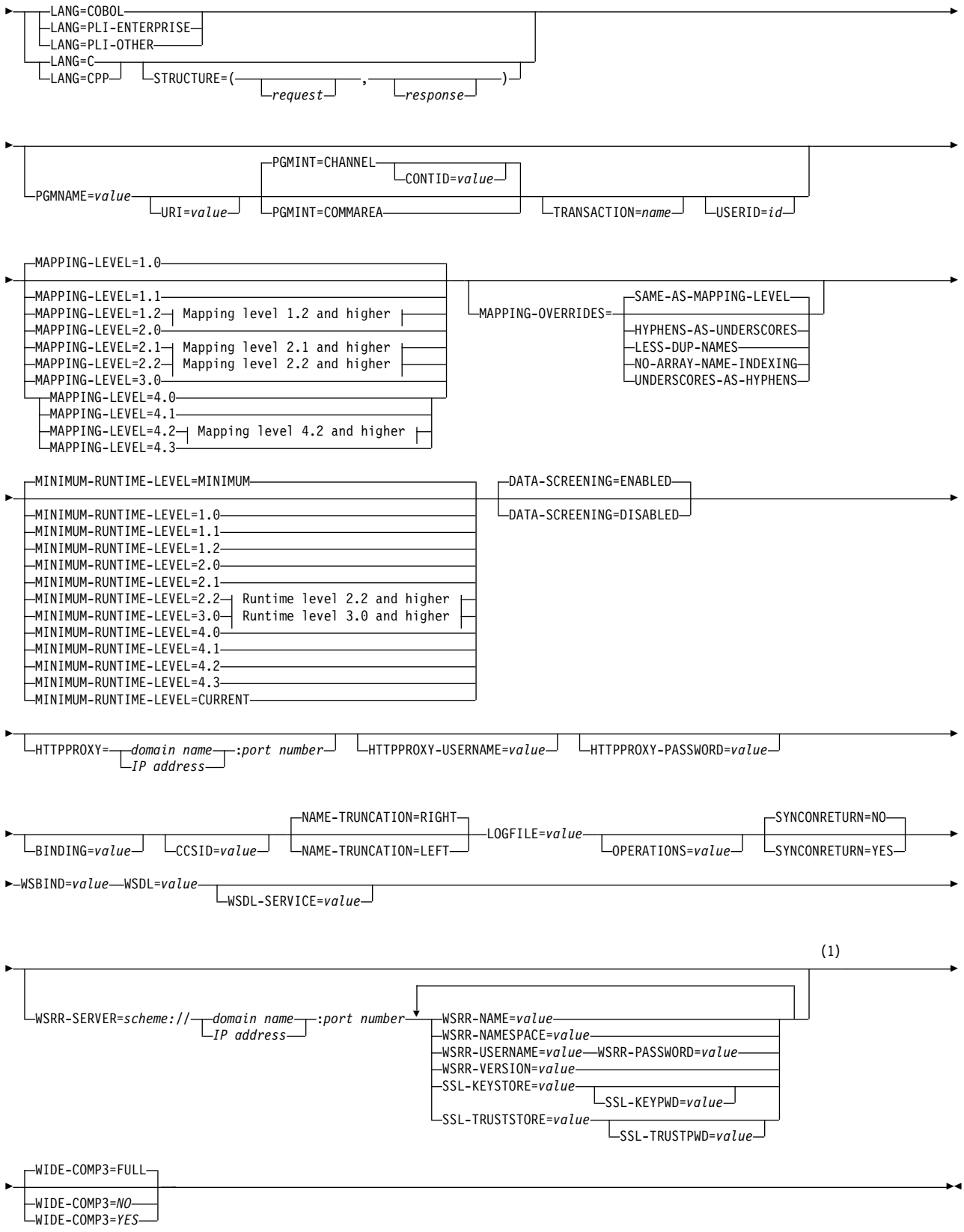
/tmp/WS2LS.err

重要: DFHWS2LS は、z/OS UNIX ファイルまたはデータ・セット・メンバーへのアクセスをロックしません。したがって、DFHWS2LS の複数のインスタンスが同時に動作して、同じ一時ワークスペース・ファイルを使用する場合には、あるジョブがワークスペース・ファイルを使っているときに別のジョブがそれを上書きするのを防ぐことはできず、予測不能な障害が発生します。

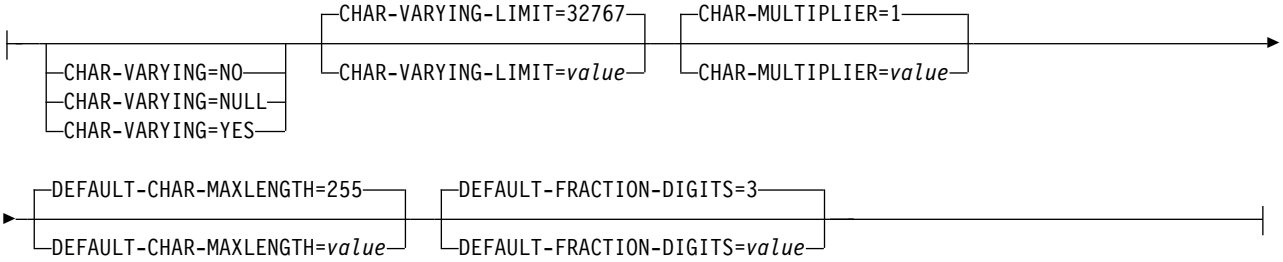
したがって、この状況を回避する命名規則および操作手順を考案することをお勧めします。例えば、システム・シンボリック・パラメーター **SYSUID** を使用すると、個々のユーザーに対して一意のワークスペース・ファイルを生成できます。これらの一時ファイルはジョブの終わりの前に削除されます。

DFHWS2LS のパラメーターの入力

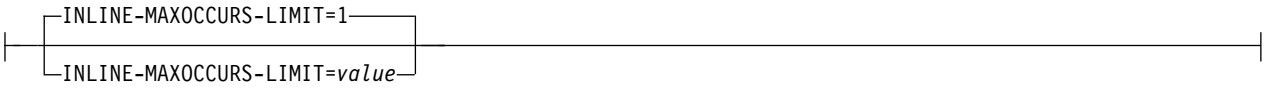



LANG=COBOL
LANG=PLI-ENTERPRISE
LANG=PLI-OTHER
LANG=C
LANG=CPP
STRUCTURE=(request , response)
PGMNAME=value
URI=value
PGMINT=CHANNEL
CONTID=value
PGMINT=COMMAREA
TRANSACTION=name
USERID=id
MAPPING-LEVEL=1.0
MAPPING-LEVEL=1.1
MAPPING-LEVEL=1.2
Mapping level 1.2 and higher
MAPPING-LEVEL=2.0
MAPPING-LEVEL=2.1
Mapping level 2.1 and higher
MAPPING-LEVEL=2.2
Mapping level 2.2 and higher
MAPPING-LEVEL=3.0
MAPPING-LEVEL=4.0
MAPPING-LEVEL=4.1
MAPPING-LEVEL=4.2
Mapping level 4.2 and higher
MAPPING-LEVEL=4.3
MAPPING-OVERRIDES=
SAME-AS-MAPPING-LEVEL
HYPHENS-AS-UNDERSCORES
LESS-DUP-NAMES
NO-ARRAY-NAME-INDEXING
UNDERSCORES-AS-HYPHENS
MINIMUM-RUNTIME-LEVEL=MINIMUM
MINIMUM-RUNTIME-LEVEL=1.0
MINIMUM-RUNTIME-LEVEL=1.1
MINIMUM-RUNTIME-LEVEL=1.2
MINIMUM-RUNTIME-LEVEL=2.0
MINIMUM-RUNTIME-LEVEL=2.1
MINIMUM-RUNTIME-LEVEL=2.2
Runtime level 2.2 and higher
MINIMUM-RUNTIME-LEVEL=3.0
Runtime level 3.0 and higher
MINIMUM-RUNTIME-LEVEL=4.0
MINIMUM-RUNTIME-LEVEL=4.1
MINIMUM-RUNTIME-LEVEL=4.2
MINIMUM-RUNTIME-LEVEL=4.3
MINIMUM-RUNTIME-LEVEL=CURRENT
DATA-SCREENING=ENABLED
DATA-SCREENING=DISABLED
HTTPPROXY=
domain name
IP address
:port number
HTTPPROXY-USERNAME=value
HTTPPROXY-PASSWORD=value
BINDING=value
CCSID=value
NAME-TRUNCATION=RIGHT
NAME-TRUNCATION=LEFT
LOGFILE=value
OPERATIONS=value
SYNCONRETURN=NO
SYNCONRETURN=YES
WSBIND=value
WSDL=value
WSDL-SERVICE=value
(1)
WSRR-SERVER=scheme://
domain name
IP address
:port number
WSRR-NAME=value
WSRR-NAMESPACE=value
WSRR-USERNAME=value
WSRR-PASSWORD=value
WSRR-VERSION=value
SSL-KEYSTORE=value
SSL-KEYPWD=value
SSL-TRUSTSTORE=value
SSL-TRUSTPWD=value
WIDE-COMP3=FULL
WIDE-COMP3=NO
WIDE-COMP3=YES

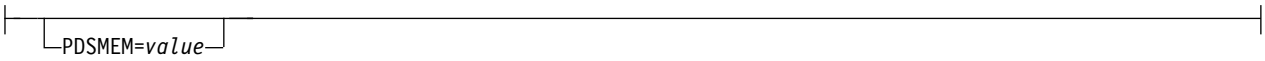
マッピング・レベル **1.2** 以上の場合:



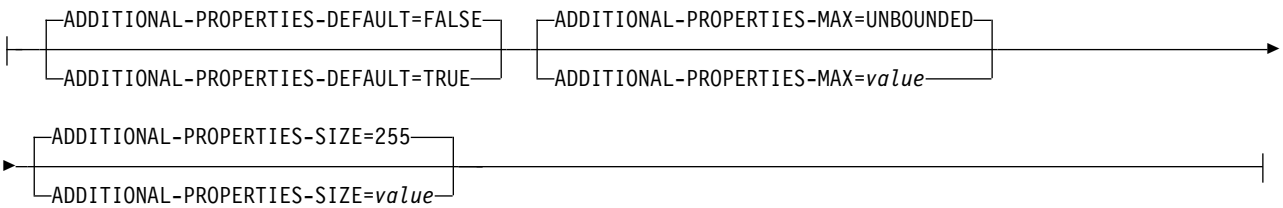
マッピング・レベル **2.1** 以上の場合:



マッピング・レベル **2.2** 以上:

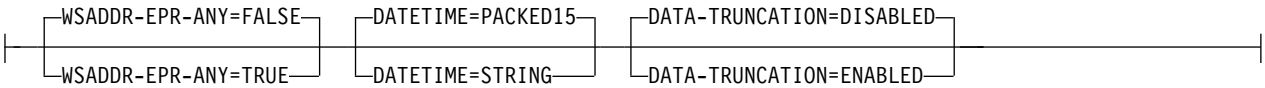


マッピング・レベル **4.2** 以上の場合:



実行時レベル **2.1** 以上:

実行時レベル **3.0** 以上:



注:

1 **WSRR-SERVER** パラメーターの設定時に指定できるそれぞれの WSRR パラメーターは、一度だけ指定可能です。

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。
- パラメーターおよび使用する場合は継続文字は、72 列を超えてはなりません。列 73 から 80 はブランクにする必要があります。
- 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。 スペースを含むアスタリスクより前の文字はすべてパラメーターの一部とみなされます。 以下に例を示します。

```
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。
- 行の末尾の文字の位置にコンマがある場合、これはオプションの行分離文字であり、無視されます。

パラメーターの記述

ADDITIONAL-PROPERTIES-DEFAULT = { **true** | **false** }

追加プロパティーのサポートを明示的に宣言していない JSON スキーマ・オブジェクトが、それらのプロパティーをサポートしていると解釈されるかどうかを示します。追加の JSON プロパティーは、JSON スキーマ内で事前に定義されていない JSON オブジェクト内のプロパティーです。これらのプロパティーは、通常、予期しない追加データとしてデータ変換メカニズムによって拒否されます。**ADDITIONAL-PROPERTIES-DEFAULT** が TRUE に設定されている場合、または JSON スキーマによってオブジェクトに対して additionalProperties:true が明示的に設定されている場合、生成されるコピーブックにこれらの値を保持するためのスペースが割り当てられます。アプリケーションは、コピーブック内の関連するフィールドを使用して、それらの値と対話できます。

このパラメーターは、マッピング・レベル 4.2 以上で使用できます。

ADDITIONAL-PROPERTIES-MAX = { **0-20** | **UNBOUNDED** }

追加プロパティーをサポートする JSON オブジェクトに対してサポートされるこれらのプロパティーの数を示します。**ADDITIONAL-PROPERTIES-DEFAULT** を参照してください。生成されるコピーブックには、追加プロパティーのアドレッシングに適した構造が含まれます。デフォルトでは、サポートされるプロパティーの数に最大制約はありません。コピーブックは、制約のない配列と同様の方法で生成され、コンテナーを使用します。このパラメーターを **INLINE-MAXOCCURS-LIMIT** パラメーターと組み合わせて使用可能な最大制約を適用し、最大数のプロパティー用に固定長の配列を割り当てることができます。これにより、コンテナーは不用になります。

このパラメーターは、マッピング・レベル 4.2 以上で使用できます。

ADDITIONAL-PROPERTIES-SIZE = { **16-32767** | **255** }

各 JSON 追加プロパティーの最大サイズを示します。JSON オブジェクトが

ADDITIONAL-PROPERTIES-DEFAULT によって定義された追加プロパティーをサポートする場合、生成されるコピーブックには、**ADDITIONAL-PROPERTIES-MAX** で指定された数までプロパティーをサポートするバインディングが設定されます。デフォルトでは、各追加プロパティーでサポートされる最大値は 255 文字です。そのサイズのフィールドは、生成されるコピーブックに生成されます。このサイズは、**ADDITIONAL-PROPERTIES-SIZE** パラメーターを設定することでカスタマイズできます。例えば、JSON オブジェクトは、以下のプロパティーを含むように処理されます。

"example": { "notes": "this extra property was not defined in the JSON Schema" }

追加プロパティーをサポートするようにコピーブックが生成されている場合は、その値全体がアプリケーションに渡されて処理されます。この値は、プロパティーのキーの前の先頭引用符で始まり、プロパティーの値の末尾の右中括弧で終わります。この例では約 100 文字です。**ADDITIONAL-PROPERTIES-SIZE** に使用する値は、考えられる最大の値を保持できる大きさにする必要があります。処理される値に対して割り当てられたバッファーが小さすぎると、エラー応答が生成されます。

このパラメーターは、マッピング・レベル 4.2 以上で使用できます。

BINDING = *value*

Web サービス記述に複数の <wsdl: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 サービス記述内の <wsdl:Binding> エレメントで使用される name 属性の値を指定します。

CCSID = *value*

アプリケーション・データ構造に文字データをエンコードするために、実行時に使用される CCSID を指定します。このパラメーターの値は、**LOCALCCSID** システム初期設定パラメーターの値を指定変更します。 *value* は、Java および z/OS 変換サービスによってサポートされている EBCDIC CCSID である必要があります (z/OS Unicode Services ユーザーズ・ガイドおよび解説書を参照)。このパラメーターを指定しない場合、アプリケーション・データ構造は、システム初期設定パラメーターで指定された CCSID を使用してエンコードされます。

このパラメーターは任意のマッピング・レベルで使用できます。

CHAR-MULTIPLIER = { **1** | *value* }

マッピング・レベルが 1.2 以降のとき、各文字に許可されるバイト数を指定します。このパラメーターの *value* は、1 から 2,147,483,647 の範囲の正整数です。 非数字に基づくマッピングはすべて、この乗数の対象です。バイナリー、数値、ゾーンおよびパック 10 進数フィールドは、この乗数の対象ではありません。

このパラメーターが役に立つのは、例えば、実行時にすべての 2 バイト文字の周囲に潜在的なシフトアウト文字とシフトイン文字のスペースを許可するために、乗数 3 を選択できる DBCS 文字の使用を計画している場合などです。

(UTF-16 を示す) **CCSID=1200** を設定する場合、**CHAR-MULTIPLIER** の有効値は 2 または 4 のみです。 UTF-16 を使用する場合のデフォルト値は 2 です。1 つ

の UTF-16 エンコード・ユニットを必要とする文字がアプリケーション・データに含まれると予期される場合は、**CHAR-MULTIPLIER=2** を使用してください。 2 つの UTF-16 エンコード・ユニットを必要とする文字がアプリケーション・データに含まれると予期される場合は、**CHAR-MULTIPLIER=4** を使用してください。

注: **CHAR-MULTIPLIER** を 1 に設定した場合、DBCS 文字は使用不可にはならず、2 に設定した場合、UTF-16 代理ペアは使用不可にはなりません。ただし、ワイド文字が定期的に使用される場合、いくつかの有効値は、割り振られたフィールドに入らなくなります。より大きな **CHAR-MULTIPLIER** 値を使用すると、XML で有効な文字数よりも多くの文字を、割り振られたフィールドに格納できます。該当する範囲制限を守るように注意する必要があります。

CHAR-VARYING = { **NO** | **NULL** | **YES** }

マッピング・レベルが 1.2 以上のとき、可変長文字データがマップされる方法を指定します。 可変長バイナリー・データ・タイプは、常にコンテナーまたは可変構造のいずれかにマップされます。 このパラメーターを指定しない場合は、指定される言語に応じてデフォルトのマッピングが異なります。 以下のオプションを選択できます。

NO 可変長文字データは固定長ストリングとしてマップされます。

NULL
可変長文字データはヌル終了ストリングにマップされます。

YES 可変長文字データは PL/I では CHAR VARYING データ・タイプにマップされます。COBOL、C および C++ 言語では、可変長文字データは、2 つの関連エレメント (データ長およびデータ) から構成される同等の表現にマップされます。

CHAR-VARYING-LIMIT = { **32767** | *value* }

マッピング・レベルが 1.2 以上のとき、言語構造にマップされるバイナリー・データおよび可変長文字データの最大サイズを指定します。 文字データまたはバイナリー・データが、このパラメーターで指定される値より大きい場合は、コンテナーにマップされ、コンテナー名が生成された言語構造で使用されます。値の範囲は 0 からデフォルトの 32,767 バイトまでです。

CONTID = *value*

サービス・プロバイダーで、SOAP メッセージを表示するために使用される最上位のデータ構造を格納するコンテナーの名前を指定します。

CICS がターゲット・アプリケーション・プログラムに渡すコンテナーの長さは、要求コンテナーの長さおよび応答コンテナーの長さより大きくなります。

DATA-SCREENING = { **ENABLED** | **DISABLED** }

アプリケーション提供のデータのエラーを検査するかどうかを指定します。

ENABLED
言語構造と矛盾するアプリケーション提供のランタイム・データは、エラーとして扱われ、メッセージ DFHPI1010 が発行されます。エラー応答がアプリケーションに 返されます。

DISABLED
言語構造と矛盾するアプリケーション提供のランタイム・データの値は、 デフォルト値で置き換えられます。例えば、数値フィールド内のゼロは誤った値を置き換えます。メッセージ DFHPI1010 は発行され

ず、通常の応答がアプリケーションに返されます。この機能を使用して、 初期設定されていない出力フィールドから生成される INVALID_PACKED_DEC および INVALID_ZONED_DEC エラー応答を回避できます。

DATA-TRUNCATION = { **DISABLED** | **ENABLED** }
固定長フィールド構造で可変長データを許容するかどうかを指定します。

DISABLED
データが CICS が予期する固定長より短い場合に、CICS は切り捨てられたデータを拒否してエラー・メッセージを発行します。

ENABLED
データが CICS が予期する固定長より短い場合に、CICS は切り捨てられたデータを許容して、欠落データをヌル値として処理します。

DATETIME = { **PACKED15** | **STRING** }
<xsd:dateTime> エレメントを言語構造にマップする方法を指定します。

PACKED15
デフォルトでは、すべての <xsd:dateTime> エレメントがタイム・スタンプとして処理され、CICS ABSTIME 形式にマップされます。

STRING
<xsd:dateTime> エレメントはテキストとして処理されます。

DEFAULT-CHAR-MAXLENGTH = { **255** | *value* }
マッピング・レベルが 1.2 以上のとき、Web サービス記述文書に長さが暗黙指定されていない場合に、マッピングについて文字データのデフォルトの配列長を文字数で指定します。 このパラメーターの value は、1 から 2,147,483,647 の範囲の正整数です。

DEFAULT-FRACTION-DIGITS = { **3** | *value* }
XML 10 進スキーマ・タイプで使用するデフォルトの小数桁数を指定します。デフォルトは 3 です。 COBOL の場合、有効な範囲は 0 から 17 までですが、パラメーター **WIDE-COMP3** が使用される場合は 0 から 30 までです。 C または PLI の場合、有効な範囲は 0 から 30 までです。

HTTPPROXY = **{** *domain name* **:** *port number* | *IP 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。 以下に例を示します。

```
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。

HTTPPROXY-PASSWORD = *value*
HTTP プロキシー・パスワードを指定します。 DFHWS2LS を実行するシステムが HTTP プロキシー・サーバーを使ってインターネットにアクセスする場合、HTTP プロキシー・サーバーが基本認証を使用するならば、**HTTPPROXY-USERNAME** と共にこのパスワードを使用する必要があります。 **HTTPPROXY** も指定した場合にのみ、このパラメーターを使用できます。

HTTPPROXY-USERNAME = *value*

HTTP プロキシー・ユーザー名を指定します。 DFHWS2LS を実行するシステムが HTTP プロキシー・サーバーを使ってインターネットにアクセスする場合、HTTP プロキシー・サーバーが基本認証を使用するならば、**HTTPPROXY-PASSWORD** と共にこのユーザー名を使用する必要があります。 **HTTPPROXY** も指定した場合にのみ、このパラメーターを使用できます。

INLINE-MAXOCCURS-LIMIT = { **1** | *value* }

maxOccurs 属性に基づいて、インラインの可変コンテンツを繰り返し使用するかどうかを指定します。 インラインにマップされる可変の繰り返しコンテンツは、生成される言語構造の残りの部分と共に現在のコンテナーに入ります。 可変の繰り返しコンテンツは、2 つの部分 (データの出現回数を格納するカウンターと、データのそれぞれの出現を格納する配列) に分けて格納されます。 可変繰り返しコンテンツに代わるマッピングとして、コンテナー・ベースのマッピングがあります。この場合、データの出現回数、およびデータを収容するコンテナーの名前を格納します。別個のコンテナーにデータを格納するとパフォーマンスに影響を与えるため、インライン・マッピングの方が適しているかもしれません。

INLINE-MAXOCCURS-LIMIT パラメーターは、マッピング・レベル 2.1 以降でのみ使用できます。 **INLINE-MAXOCCURS-LIMIT** の値は、 0 から 32,767 の範囲の正整数です。 値 0 は、インライン・マッピングを使用しないことを示します。 値 1 を使用すると、オプション・エレメントがインラインでマップされます。 maxOccurs 属性の *value* が **INLINE-MAXOCCURS-LIMIT** の *value* より大きい場合、コンテナーに基づくマッピングが使用されます。それ以外の場合はインライン・マッピングが使用されます。

可変の繰り返しリストをインラインでマップするかどうか決定する際には、繰り返されるデータの単一項目の長さを考慮してください。長い項目が少ない回数だけ出現する場合は、コンテナーに基づくマッピングが適しています。短い項目が数多く出現する場合は、インライン・マッピングが適しています。

LANG = **COBOL**|**PLI-ENTERPRISE**|**PLI-OTHER**|**C**|**CPP**

高水準言語構造のプログラミング言語を指定します。

COBOL
COBOL

PLI-ENTERPRISE
Enterprise PL/I

PLI-OTHER
Enterprise PL/I 以外の PL/I のレベル

C C

CPP C++

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 z/OS UNIX 名です。DFHWS2LS は、このファイルが存在しない場合、ディレクトリー構造は作成しませんがファイルを作成します。

通常はこのファイルを使用しませんが、DFHWS2LS 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。

MAPPING-LEVEL = { **1.0** | **1.1** | **1.2** | **2.0** | **2.1** | **2.2** | **3.0** | **4.0** | **4.1** | **4.2** | **4.3** }

Web サービス・バインディング・ファイルおよび言語構造を生成する際に、DFHWS2LS が使用するマッピングのレベルを指定します。 以下のオプションを選択できます。

1.0 このマッピング・レベルはデフォルトです。 Web サービス・バインディング・ファイルおよび言語構造は CICS TS 3.1 マッピング・レベルを使用して生成されます。

1.1 XML 属性や <list> および <union> データ・タイプは言語構造にマップされます。 最大長が 32,767 バイトより大きい文字データおよびバイナリー・データはコンテナーにマップされます。 コンテナー名は言語構造内に作成されます。

1.2 **CHAR-VARYING** および **CHAR-VARYING-LIMIT** パラメーターを使用して、実行時に文字データがマップおよび処理される方法を制御します。このパラメーターのどちらも指定しない場合、最大長が 32,768 バイトより小さいバイナリー・データおよび文字データは、C++ を除くすべての言語で VARYING 構造にマップされます。C++ では、文字データはヌル終了ストリングにマップされます。

2.0 このマッピング・レベルは、CICS TS 3.2 領域またはそれ以降で、言語構造と Web サービス・バインディング・ファイル間のマッピングに対する機能拡張を利用するときに使用します。

2.1 このマッピング・レベルを CICS TS 3.2 以降の領域で使用すると、<xsd:any> および xsd:anyType がサポートされることに加えて、**INLINE-MAXOCCURS-LIMIT** パラメーターを使って可変の繰り返しコンテンツをインラインでマップするオプションを使用でき、<xsd:sequence>、<xsd:choice>、<xsd:all> で minOccurs="0" がサポートされます。

2.2 このマッピング・レベルは、CICS TS 3.2 以降の領域で、以下のサポートを利用するために使用します。

- 固定値を持つエレメント
- <xsd:choice> エレメントの拡張サポート
- 抽象データ・タイプ
- 抽象エレメント
- 置換グループ

3.0 このマッピング・レベルは、CICS TS 4.1 以降の領域で使用します。このマッピング・レベルでは、タイム・スタンプを CICS ABSTIME 形式に変換することができます。

4.0 UTF-16 を使用する場合には、CICS TS 5.2 以降の領域でこのマッピング・レベルを使用します。

4.1 切り捨て可能な配列をサポートするには、CICS TS 5.2 以降の領域でこのマッピング・レベルを使用します。

4.2 このマッピング・レベルは、CICS TS 5.4 以降の領域で使用します。

4.3 このマッピング・レベルは、CICS TS 5.4 以降の領域で使用します。

マッピング・レベルについて詳しくは、CICS アシスタントのマッピング・レベルを参照してください。

MAPPING-OVERRIDES = { SAME-AS-MAPPING-LEVEL | HYPHENS-AS-UNDERSCORES | LESS-DUP-NAMES | NO-ARRAY-NAME-INDEXING | UNDERSCORES-AS-HYPHENS }

言語構造を生成するときの指定されたマッピング・レベルについて、デフォルトの動作を指定変更するかどうかを指定します。

注: 任意のサブオプションをコンマ区切りリストで使用できます。これらのオプションは相互に排他的ではありません。組み合わされて、順不同で使用されます。

SAME-AS-MAPPING-LEVEL

このパラメーターは、マッピング・レベルと同じスタイルで言語構造を生成します。これはデフォルトです。

HYPHENS-AS-UNDERSCORES

PL/I のみ。このパラメーターは、生成される PL/I 言語構造を読みやすくするために、WSDL 文書内のすべてのハイフンを文字 X ではなく下線に変換します。詳しくは、XML スキーマから PL/I へのマッピングを参照してください。このオプションは、マッピング・レベル 4.2 で自動的に有効になります。

LESS-DUP-NAMES

このパラメーターは、フィールドを直接参照できるように名前の末尾に _value を付けた非構造性の構造フィールド名を生成します。例えば、以下の PL/I 言語構造で、MAPPING-OVERRIDES=LESS-DUP-NAMES が指定されるとき、レベル 12 フィールドの streetName には接尾部に _value が付いています。

```
09 streetName,
12 streetName CHAR(255) VARYING
UNALIGNED,
12 filler BIT (7),
12 attr_nil_streetName_value BIT (1),
```

結果構造は以下のとおりです。

```
09 streetName,
12 streetName_value CHAR(255) VARYING
UNALIGNED,
12 filler BIT (7),
12 attr_nil_streetName_value BIT (1),
```

このオプションは、マッピング・レベル 4.2 で自動的に有効になります。

NO-ARRAY-NAME-INDEXING

COBOL および Enterprise PL/I のみ。配列内のフィールド名が、構造の上位の範囲内でのみ固有であることを確認してください。

UNDERSCORES-AS-HYPHENS

このオプションは、マッピング・レベル 4.0 で自動的に有効になります。

COBOL のみ。 このパラメーターは、WSDL 文書内のすべての下線を文字 X ではなくハイフンに変換して、生成される COBOL 言語構造の

読みやすさを向上します。フィールド名の競合が発生する場合、そのフィールドが必ず固有になるように番号が付きます。 詳しくは、XML スキーマとCOBOL のマッピングを参照してください。

MINIMUM-RUNTIME-LEVEL = { MINIMUM | 1.0 | 1.1 | 1.2 | 2.0 | 2.1 | 2.2 | 3.0 | 4.0 | 4.1 | 4.2 | 4.3 | CURRENT }

Web サービス・バインディング・ファイルを配置できる最小の CICS 実行時環境を指定します。指定した他のパラメーターと一致しないレベルを選択すると、エラー・メッセージを受け取ります。以下のオプションを選択できます。

MINIMUM
選択したパラメーターを前提として、最低限可能な CICS 実行時レベルが自動的に割り振られます。

1.0 生成された Web サービス・バインディング・ファイルが、CICS TS 3.1 領域で正常に配置されます。一部のパラメーターは、このランタイム・レベルでは使用できません。

1.1 生成された Web サービス・バインディング・ファイルが、CICS TS 3.1 領域で正常に配置されます。**MAPPING-LEVEL** パラメーターには、マッピング・レベル 1.1 以前を使用できます。一部のパラメーターは、このランタイム・レベルでは使用できません。

1.2 生成された Web サービス・バインディング・ファイルが、CICS TS 3.1 領域で正常に配置されます。**MAPPING-LEVEL** パラメーターには、マッピング・レベル 1.2 以前を使用できます。一部のパラメーターは、このランタイム・レベルでは使用できません。

2.0 生成された Web サービス・バインディング・ファイルが、CICS TS 3.2 以降の領域で正常に配置されます。 **MAPPING-LEVEL** パラメーターには、マッピング・レベル 2.0 以前を使用できます。一部のパラメーターは、このランタイム・レベルでは使用できません。

2.1 生成された Web サービス・バインディング・ファイルは、CICS TS 3.2 以降の領域に正常にデプロイされます。**MAPPING-LEVEL** パラメーターには、マッピング・レベル 2.1 以前を使用できます。一部のパラメーターは、このランタイム・レベルでは使用できません。

2.2 生成された Web サービス・バインディング・ファイルは、CICS TS 3.2 以降の領域に正常にデプロイされます。この実行時レベルでは、**MAPPING-LEVEL** パラメーターには、マッピング・レベル 2.2 以前を使用することができます。一部のパラメーターは、このランタイム・レベルでは使用できません。

3.0 生成された Web サービス・バインディング・ファイルは、CICS TS 4.1 以降の領域に正常にデプロイされます。この実行時レベルでは、**MAPPING-LEVEL** パラメーターには、マッピング・レベル 3.0 以前を使用することができます。一部のパラメーターは、このランタイム・レベルでは使用できません。

4.0 生成された Web サービス・バインディング・ファイルは、CICS TS 5.2 以降の領域に正常にデプロイされます。この実行時レベルでは、

MAPPING-LEVEL パラメーターにマッピング・レベル 4.0 以前を使用できます。このレベルでは任意のオプション・パラメーターを使用できます。

4.1 生成された Web サービス・バインディング・ファイルは、CICS TS 5.2 以降の領域に正常にデプロイされます。この実行時レベルでは、**MAPPING-LEVEL** パラメーターには、マッピング・レベル 4.1 以前を使用することができます。

4.2 生成された Web サービス・バインディング・ファイルは、CICS TS V5.4 以降の領域に正常にデプロイされます。この実行時レベルでは、**MAPPING-LEVEL** パラメーターには、マッピング・レベル 4.2 以前を使用することができます。

4.3 生成された Web サービス・バインディング・ファイルは、CICS TS 5.4 以降の領域に正常にデプロイされます。この実行時レベルでは、**MAPPING-LEVEL** パラメーターには、マッピング・レベル 4.3 以前を使用することができます。

CURRENT
生成された Web サービス・バインディング・ファイルは、Web サービス・バインディング・ファイルの生成に使用するものと同じ実行時レベルで、CICS 領域に正常に配置されます。

NAME-TRUNCATION = { LEFT | RIGHT }
XML エレメント名が左と右のどちらから切り捨てられるかを指定します。CICS Web サービス・アシスタントは、指定された高水準言語における適切な長さに XML エレメント名を切り捨てます。デフォルトでは、名前は右から切り捨てられます。

OPERATIONS = *value*
Web サービス・リクエスター・アプリケーションについて、Web サービス・バインディング・ファイルを生成するために使用される Web サービス記述から、有効な <wsdl:Operation> エレメントのサブセットを指定します。 各 <wsdl:Operation> エレメントはスペースで分離します。必要に応じて、リストは複数行にわたることがあります。 このパラメーターは WSDL 1.1 文書および WSDL 2.0 文書の両方で使用できます。

PDSCP = *value*
REQMEM および **RESPMEM** パラメーターに指定される区分データ・セット・メンバーで使用されるコード・ページを指定します。ここで、*value* は CCSID 番号または Java コード・ページ番号です。このパラメーターを指定しない場合は、z/OS UNIX システム・サービスのコード・ページが使用されます。 例えば、**PDSCP**=037 と指定できます。

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。

PDSMEM = *value*
以下に示す抽象的なデータ・タイプ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。 メンバー名は、接頭部に 2 桁の数値を付加することで生成されます。

このパラメーターは、抽象データ・タイプに関連した言語構造の名前を指定するために、マッピング・レベル 2.2 以上で使用します。 **PDSMEM** パラメーターを省略した場合、抽象データ・タイプの言語構造の名前は **REQMEM** パラメーターの値を使って指定されます。

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。

ターゲット・アプリケーション・プログラムが要求を処理するとき、同じメカニズムを使用して応答を戻す必要があります。要求が通信域で受信されている場合、応答を通信域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、応答をコンテナーに戻す必要があります。CICS がターゲット・アプリケーション・プログラムに渡す通信域またはコンテナーの長さは、要求の通信域またはコンテナーの長さ、および応答の通信域またはコンテナーの長さより大きくなります。

PGMNAME = *value*

CICS PROGRAM リソース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・プログラムのリソース名を指定します。

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します。

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・プログラムへの入力です。
- 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。メンバー名は、接頭部に 2 桁の数値を付加することで生成されます。

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。メンバー名は、接頭部に 2 桁の数値を付加することで生成されます。

応答が存在しない場合 (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。

SSL-KEYSTORE = *value*

このオプション・パラメーターは、鍵ストア・ファイルの完全修飾された場所を指定します。

Web サービス・アシスタントが SSL (Secure Sockets Layer) 暗号化を使用して、ネットワークを介して IBM WebSphere Service Registry and Repository (WSRR) と通信できるようにする場合、このパラメーターを使用します。

SSL-KEYPWD = *value*

このオプション・パラメーターは、鍵ストアのパスワードを指定します。

Web サービス・アシスタントが SSL (Secure Sockets Layer) 暗号化を使用して、ネットワークを介して IBM WebSphere Service Registry and Repository (WSRR) と通信できるようにする場合、このパラメーターを使用します。

SSL-TRUSTSTORE = *value*

このオプション・パラメーターは、トラストストア・ファイルの完全修飾された場所を指定します。

Web サービス・アシスタントが SSL (Secure Sockets Layer) 暗号化を使用して、ネットワークを介して IBM WebSphere Service Registry and Repository (WSRR) と通信できるようにする場合、このパラメーターを使用します。

SSL-TRUSTPWD = *value*

このオプション・パラメーターは、トラストストアのパスワードを指定します。

Web サービス・アシスタントが SSL (Secure Sockets Layer) 暗号化を使用して、ネットワークを介して IBM WebSphere Service Registry and Repository (WSRR) と通信できるようにする場合、このパラメーターを使用します。

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、 *request nn* および *response nn* 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。

SYNCONRETURN = { **NO** | **YES** }
リモート Web サービスが同期点を発行できるかどうかを指定します。

NO リモート Web サービスは同期点を発行できません。 この値はデフォルトです。リモート Web サービスが同期点を発行すると、ADPL 異常終了になり失敗します。

YES リモート Web サービスは同期点を発行できます。 YES を選択した場合、リモート Web サービスから制御が戻されたときにリモート・タスクが別個の作業単位としてコミットされます。リモート Web サービスがリカバリー可能リソースを更新して戻した後に障害が発生した場合、そのリソースの更新内容をバックアウトすることはできません。

TRANSACTION = *name*
サービス・プロバイダーで、このパラメーターは、応答を組み立てるためにパイプラインを開始できる 1 から 4 文字の別名トランザクションの名前を指定します。 このパラメーターの値は、URIMAP リソースが **PIPELINE** スキャン・コマンドを使用して自動的に作成される際に、URIMAP リソースの TRANSACTION 属性を定義するために使用されます。

許容文字:
A-Z a-z 0-9 $ @ # _ < >

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 のパスのコンポーネントを指定します。

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。 CICS は、サービス・リクエスターの URIMAP リソースは生成しません。 ターゲット Web サービスの URI へのクライアント要求を行うときに、サービス・リクエスターが使用するための独自の URIMAP リソースを定義できます。 サービス・リクエスターが **INVOKE SERVICE** コマンドを発行すると、CICS は、Web サービス定義に指定されている wsdl:port の soap:address の場所を使用します (存在する場合)。これを指定変更し、**INVOKE SERVICE** コマンドで URIMAP または URI オプションを使用して別の URI を指定できます。

USERID = *id*
サービス・プロバイダーでは、このパラメーターは、任意の Web クライアントで使用可能な 1 文字から 8 文字のユーザー ID を指定します。 アプリケーションから生成される応答や Web サービスについては、そのユーザー ID の下で別名トランザクションが追加されます。このパラメーターの値は、URIMAP リソースが **PIPELINE** スキャン・コマンドを使用して自動的に作成される際に、URIMAP リソースの USERID 属性を定義するために使用されます。

許容文字:

A-Z a-z 0-9 $ @ #

WIDE-COMP3 = { **FULL** | **NO** | **YES** }

生成される COBOL または PL/I 言語構造でパック 10 進数の可変長の最大サイズを制御します。

FULL COBOL および PL/I の場合。DFHJS2LS は、すべての有効な値を保持できるサイズでパック 10 進数フィールドを生成します。最大サイズは 31 桁です。これはデフォルトです。

NO COBOL のみ。 DFHJS2LS は、COBOL 言語構造タイプ COMP-3 を生成しているとき、パック 10 進数の可変長を 18 に制限します。 パック 10 進数のサイズが 18 より大きい場合、指定されたタイプは合計 18 桁に制限されていることを示すメッセージ DFHPI9022W が出されます。

YES COBOL のみ。 DFHJS2LS は、COBOL 言語構造タイプ COMP-3 を生成しているとき、最大サイズ 31 をサポートします。

注: NO および YES オプションを指定すると、すべての有効な値を表わすことができないフィールドが生成されます。FULL オプションは、この問題を回避します。ただし、FULL オプションを指定すると、パック 10 進数フィールドで無効な値が表される可能性があります。例えば、スキーマに最大サイズとして最大 5 桁の数字と最大 2 桁の小数点以下の数値が指定されている場合、FULL オプションを指定すると、7 桁の数字が許容されるパック 10 進数フィールドが生成されます。この場合、25000 や 999.99 などの有効な値を収容するだけのスペースがあると同時に、9999.99 などの無効な値を収容するだけのスペースも提供されることになります。FULL オプションを使用する場合は、アプリケーション・データに無効な値が生成されないよう注意してください。

WSADDR-EPR-ANY = { **TRUE** | **FALSE** }

CICS で 1 つの WS-Addressing エンドポイント参照 (EPR) を言語構造のコンポーネント部分に変換するか、または EPR を 1 つの <xsd:any> タイプとして扱うかを指定します。 EPR を 1 つの <xsd:any> タイプとして扱う場合、**WSACONTEXT BUILD** API は EPR XML を直接使用できます。

FALSE

DFHWS2LS は通常どおりに動作し、XML を高水準言語構造に変換します。

TRUE このオプションを TRUE に設定すると、ランタイム CICS は EPR 全体を <xsd:any> タイプとして扱い、 EPR XML はアプリケーションが参照できるコンテナーに入れられます。 アプリケーションは EPR XML を **WSACONTEXT BUILD** API と共に使用して、アドレッシング・コンテキストで EPR を構成できます。

このパラメーターは実行時レベル 3.0 以降でのみ使用できます。

WSBIND = *value*

Web サービス・バインディング・ファイルの完全修飾 z/OS UNIX 名です。

DFHWS2LS は、このファイルが存在しない場合、ディレクトリー構造は作成しませんがファイルを作成します。 デフォルトのファイル拡張子は .wsbind です。

WSDL = *value*
Web サービス記述を格納するファイルの完全修飾 z/OS UNIX 名です。 ファイル名に使用できる文字は、URL に有効な文字に制限され、特に # 文字を含めることはできません。WSRR を使って WSDL 文書を取り出す場合、このパラメーターは WSDL 文書のローカル・コピーの書き込み先となるファイル・システム上の場所を指定します。

WSDL-SERVICE = *value*
Web サービス記述に Binding エレメントについて複数の Service エレメントが含まれるときに使用する、wsdl:Service エレメントを指定します。 **BINDING** パラメーターの値を指定する場合は、このパラメーターに指定する Service エレメントが、指定された Binding エレメントと整合している必要があります。このパラメーターは WSDL 1.1 文書または WSDL 2.0 文書のいずれかで使用できます。

WSRR-NAME = *value*
WSRR から取り出す WSDL 文書の名前を指定します。このパラメーターは、**WSRR-SERVER** パラメーターが指定された場合にのみ使用します。

WSRR-NAMESPACE = *value*
WSRR から取り出す WSDL 文書の名前空間を指定します。 **WSRR-SERVER** パラメーターが指定されている場合、オプションでこのパラメーターを使用して、**WSRR-NAME** パラメーターに示される WSDL 文書名を完全修飾することができます。

WSRR-PASSWORD = *value*
WSRR へのアクセスにパスワードの入力が必要な場合は、このオプション・パラメーターを使用します。

WSRR-USERNAME パラメーターを指定する場合には、このパラメーターも指定する必要があります。

このパラメーターは、**WSRR-SERVER** パラメーターが指定された場合にのみ使用します。

WSRR-SERVER = { *domain name* **:** *port number* | *IP address* **:** *port number* }
このパラメーターを使用して、IBM WebSphere Service Registry and Repository (WSRR) サーバーの場所を指定します。このパラメーターを指定した場合、WSRR パラメーター検証が使用されます。

WSRR-USERNAME = *value*
WSRR へのアクセスでユーザー名を指定する必要がある場合は、このオプション・パラメーターを使用します。 WSRR はこのユーザー名を使って所有者プロパティーを設定します。

このパラメーターは、**WSRR-SERVER** パラメーターが指定された場合にのみ使用します。

WSRR-VERSION = *value*
WSRR から取り出す WSDL 文書のバージョンを指定します。このパラメーターは、**WSRR-SERVER** パラメーターが指定されている場合に限り使用できます。

XML-ONLY = { **TRUE** | **FALSE** }

CICS で SOAP メッセージ内の XML をアプリケーション・データに変換するかどうかを指定します。 XML 自体を処理する Web サービス・アプリケーションを作成するには、**XML-ONLY** パラメーターを使用してください。

TRUE CICS は XML への変換をまったく実行しません。 サービス・リクエスターまたはプロバイダー・アプリケーションは、DFHWS-BODY コンテナーの内容を処理して XML と高水準言語の間でデータを直接マップする必要があります。

FALSE
CICS は XML を高水準言語に変換します。

このパラメーターは実行時レベル 2.1 以降でのみ使用できます。

その他の情報

- DFHLS2SC を実行するユーザー ID は、UNIX システム・サービスを使用するように構成されていなければなりません。 ユーザー ID には、CICS z/OS UNIX ファイル構造および PDS ライブラリーに対する読み取り権限、および **LOGFILE**、**WSBIND**、および **WSDL** パラメーターで指定されたディレクトリーへの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。
- JCL の最大パラメーター長は 100 文字です。 これは、**STDPARM** ステートメントを使用して増やすことができます。詳しくは、z/OS UNIX システム・サービス ユーザーズ・ガイドを参照してください。

例

```
//WS2LS JOB '
accounting information
',
name,MSGCLASS=A
// SET QT=''''
//JAVAPROG EXEC DFHWS2LS,
// TMPFILE=&QT.&SYSUID.&QT
//INPUT.SYSUT1 DD *
PDSLIB=//CICSHLQ.SDFHSAMP
REQMEM=CPYBK1
RESPMEM=CPYBK2
LANG=COBOL
LOGFILE=/u/exampleapp/wsbind/example.log
MAPPING-LEVEL=3.0
MAPPING-OVERRIDES=UNDERSCORES-AS-HYPHENS
CHAR-VARYING=NULL
INLINE-MAXOCCURS-LIMIT=2
PGMNAME=DFH0XCMN
URI=exampleApp/example
PGMINT=COMMAREA
SYNCONRETURN=YES
WSBIND=/u/exampleapp/wsbind/example.wsbind
WSDL=/u/exampleapp/wsdl/example.wsdl
/*
```

DFHWS2LS が **WS-Addressing** をサポートするために必要とするパラメーター:

WSDL を Web Services Addressing 用に構成する場合、Web サービス・アシスタントである DFHWS2LS の **MINIMUM-RUNTIME** パラメーターおよび **MAPPING-LEVEL**

パラメーターの値を 3.0 以上に設定する必要があります。**WSADDR-EPR-ANY** パラメーターを TRUE に設定することを検討することもできます。

Web サービス・アシスタント DFHWS2LS の **MINIMUM-RUNTIME** パラメーターを 3.0 以上に設定します。 3.0 以上のランタイム・レベルでは、アシスタントが生成する WSBind ファイルはすべて Web Services Addressing を完全にサポートし、他の Web サービス・プラットフォームとも確実に相互運用できます。

Web サービス・アシスタント DFHWS2LS の **MAPPING-LEVEL** パラメーターを 3.0 以上に設定します。

WSDL 文書で定義されている要求メッセージまたは応答メッセージにタイプ wsa:EndpointReferenceType のエレメントがある場合、またそれらのエレメントを **WSACONTEXT BUILD** API コマンドへの入力として実行時に使用する場合、**WSADDR-EPR-ANY** パラメーターを TRUE に設定します。 **WSADDR-EPR-ANY** パラメーターを TRUE に設定する場合、 CICS は EPR を言語構造に実行時に変換することはしません。代わりに CICS は、EPR データを <xsd:any> エレメントとして扱い、指定されたコンテナーに格納する必要があります。

以下の例の WSDL フラグメントは、タイプ wsa:EndpointReferenceType のエレメントとして渡される <wsa:To> MAP を示しています。

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="exampleEPR" targetNamespace="http://example.ibm.com/"
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:s0="http://example.ibm.com/"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:wsa="http://www.w3.org/2005/08/addressing"
    xmlns:wsam="http://www.w3.org/2007/05/addressing/metadata">
  <types>
    <xs:schema targetNamespace="http://test.org/"
            xmlns:s="http://www.w3.org/2001/XMLSchema"
            xmlns:s0="http://example.ibm.com/"
            xmlns:wsa="http://www.w3.org/2005/08/addressing">
      ...
       <xs:element name="exampleResponse" type="s0:typeResponse"/>
       <xs:complexType name="typeResponse">
         <xs:sequence>
           <xs:element name="myEpr" type="wsa:EndpointReferenceType"/> 1
         </xs:sequence>
       </xs:complexType>
       ...
    </xs:schema>
  </types>
  ...
  <message name="msgResponse">
    <part element="s0:exampleResponse" name="response"/>
  </message>
  ...
</definitions>
```

WSADDR-EPR-ANY パラメーターを TRUE に設定した DFHWS2LS によってエレメント <xs:element name="myEpr" type="wsa:EndpointReferenceType"/> 1 が処理されると、 myEpr エレメント・データは指定されたコンテナーに実行時には <xsd:any> エレメントとして格納され、生成される言語構造にそのコンテナーへのポインターとして追加されます。

例えば、myEpr エレメントに関して DFHWS2LS によって生成される COBOL 言語構造をここに示します。

```
09 myEpr.
   12 myEpr-xml-cont               PIC X(16).
   12 myEpr-xmlns-cont             PIC X(16).
```

myEpr-xml-cont コンテナーは、myEpr データを格納するコンテナーの名前を格納します。 myEpr-xmlns-cont は、スコープ内にある XML 名前空間宣言を取り込むオプションのコンテナーです。

CICS 支援機能用のマッピング・レベル

マッピングは、言語構造と XML スキーマの間で情報が変換される方法を指定するのに使用される一連の規則です。 現在使用可能で最も高度なマッピングを活用するには、CICS 支援機能の **MAPPING-LEVEL** パラメーターを最新レベルに設定する必要があります。

マッピングの各レベルでは、最高レベルのマッピングによって利用可能な最良の機能が提供される以前のマッピング機能が継承されます。一番高いマッピング・レベルでは、実行時のデータ変換をさらに制御することができ、特定のデータ・タイプおよび XML エレメントに対するサポートの制限も解除されます。

以前のレベルで有効だったアプリケーションを再展開したい場合には、**MAPPING-LEVEL** パラメーターをそのレベルに設定できます。

マッピング・レベル 4.3

マッピング・レベル 4.3 は、APAR PI88519 が適用された CICS TS V5.4 以上と互換性があります。

マッピング・レベル 4.3 は、主に DFHJS2LS で使用されますが、CICS Web サービス支援機能、XML 支援機能、および JSON 支援機能にも含まれています。このマッピング・レベルは、JSON での多次元配列のサポートを実装します。

マッピング・レベル 4.2

マッピング・レベル 4.2 は、APAR PI86039 が適用された CICS TS V5.4 以上と互換性があります。

マッピング・レベル 4.2 は、主に DFHJS2LS で使用されますが、CICS Web サービス支援機能、XML 支援機能、および JSON 支援機能にも含まれています。このマッピング・レベルは、JSON の追加プロパティーのサポートを実装し、DFHJS2LS に 3 つのパラメーター (**ADDITIONAL-PROPERTIES-DEFAULT**、**ADDITIONAL-PROPERTIES-MAX**、および **ADDITIONAL-PROPERTIES-SIZE**) を導入します。

マッピング・レベル 4.1

マッピング・レベル 4.1 は、CICS TS 5.3 の領域や APAR PI67641 が適用された CICS TS 5.2 の領域、およびこれら以降の領域との互換性があります。

マッピング・レベル 4.1 は、CICS Web サービス支援機能、XML 支援機能、および JSON 支援機能に追加されています。このマッピング・レベルは、既存のコピーブックからボトムアップで生成される単純配列に対して改善されたマッピングを実

装します。また、配列内で初期化されていない末尾のストレージの自動検出機能、およびそのようなレコードを生成される XML/JSON フォームから省略する機能を CICS に追加します。詳しくは、 482 ページの『COBOL から JSON スキーマへのマッピング』を参照してください。

DFHLS2WS、DFHWS2LS、DFHLS2SC、DFHSC2LS、DFHJS2LS、および DFHLS2JS は **TRUNCATE-NULL-ARRAYS** および **TRUNCATE-NULL-ARRAY-VALUES** パラメーターをサポートします。

TRUNCATE-NULL-ARRAY-VALUES に値を指定する場合、 **TRUNCATE-NULL-ARRAYS=ENABLED** も指定する必要があります。

マッピング・レベル 4.0

マッピング・レベル 4.0 は、CICS TS 5.2 領域と互換性があります。

マッピング・レベル 4.0 以上では、DFHLS2SC および DFHLS2WS は、COBOL OCCURS DEPENDING ON 節をサポートし、COBOL 文字配列の XML ストリングへのマッピングをサポートします。この動作を設定するには、CICS 支援機能で、**CHAR-OCCURS** パラメーターを使用します。

- パラメーター **DATA-TRUNCATION=ENABLED** を指定する必要があります。
- 複合 OCCURS DEPENDING ON はサポートされません。この制約は、構造体の最後のフィールドに対してのみ OCCURS DEPENDING ON がサポートされることを意味します。
- CICS は、OCCURS DEPENDING ON 節のターゲットとして修飾名 ('OF' キーワードの使用) をサポートしません (例えば、`FIELD1 OF STRUCTURE1`)。
- CICS は UNBOUNDED キーワードをサポートしません。アプリケーションによって予期される表の最大サイズを指定する必要があります。

マッピング・レベル 4.0 以上では、CICS Web サービスは、UTF-16 Unicode を使用してエンコードされるアプリケーション・データの変換をサポートします。

- LS2WS または LS2SC を使用する場合、UTF-16 に言語固有のデータ型を使用することによって、この動作を使用可能にできます。
- WS2LS または SC2LS を使用する場合、CCSID=1200 を設定することによって、この動作を使用可能にできます。
- CICS がサポートしている Unicode コード・ページは、「UTF-16BE with IBM Private Use Area」(CCSID 1200) のみです。
- UTF-8 を使用してエンコードされたアプリケーション・データの変換はサポートされていません。

注: DFHLS2WS および DFHLS2SC は、COBOL の `GROUP USAGE NATIONAL` 節をサポートしていません。

マッピング・レベル 3.0 以上

マッピング・レベル 3.0 は、CICS TS 4.1 以降の領域と互換性があります。

このマッピング・レベルでは、以下のサポートが提供されます。

- DFHSC2LS および DFHWS2LS は、xsd:dateTime データ型を CICS ASKTIME 形式にマップします。
- DFHLS2WS は、WSDL 文書および Web サービス・バインディングを、1 つのコンテナーのみでなく、多数のコンテナーを使用するアプリケーションから生成できます。
- 固定長データ構造で記述される切り捨てられたデータの許容。この動作は、CICS 支援機能で **DATA-TRUNCATION** パラメーターを使用することによって設定できます。

マッピング・レベル 2.2 以上

マッピング・レベル 2.2 は、APAR PK69738 およびそれ以上が適用された CICS TS 3.2 領域と互換性があります。

マッピング・レベル 2.2 以上では、DFHSC2LS および DFHWS2LS は次の XML マッピングをサポートします。

- エレメントの固定値
- 置換グループ
- 抽象データ・タイプ
- XML スキーマ <sequence> エレメントを <choice> エレメント内にネストできる

DFHSC2LS および DFHWS2LS は次に示す XML マッピングに対する拡張サポートを提供します。

- 抽象エレメント
- XML スキーマ <choice> エレメント

マッピング・レベル 2.1 以上の場合

マッピング・レベル 2.1 は、APAR PK59794 およびそれ以上が適用された CICS TS 3.2 領域と互換性があります。

このマッピング・レベルでは、新規の **INLINE-MAXOCCURS-LIMIT** パラメーターによって、および **CHAR-VARYING** パラメーターの新規の値によって可変コンテンツを処理する方法をさらに制御できます。

マッピング・レベル 2.1 以上では、DFHSC2LS および DFHWS2LS は XML マッピングについて以下に示す新規および改良されたサポートを提供します。

- XML スキーマ <any> エレメント
- xsd:anyType タイプ
- 抽象エレメントの許容
- **INLINE-MAXOCCURS-LIMIT** パラメーター
- minOccurs 属性

INLINE-MAXOCCURS-LIMIT パラメーターは、可変の繰り返しリストがインラインにマップされるかどうかを指定します。 インラインへの可変反復内容のマップの詳細については、変化するエレメントの配列を参照してください。

minOccurs 属性のサポートは、XML スキーマの <sequence> エレメント、<choice> エレメント、および <all> エレメントに対して拡張されています。minOccurs="0" の場合、CICS 支援機能はこれらのエレメントを、minOccurs="0" 属性もそのすべての子エレメントの属性であるかのように扱います。

マッピング・レベル 2.1 以上では、DFHLS2SC および DFHLS2WS は次の XML マッピングをサポートします。

- COBOL および PL/I での FILLER フィールドは無視されます
- **CHAR-VARYING** パラメーターに対する COLLAPSE 値
- **CHAR-VARYING** パラメーターに対する BINARY 値

COBOL および PL/I での FILLER フィールドは無視されます。生成される XML スキーマにはこれらが表示されず、相応のギャップが実行時にデータ構造に残されます。

COLLAPSE は、 CICS がテキスト・フィールドで末尾スペースを無視するように します。

BINARY は 2 進数フィールドのサポートを提供します。この値は、COBOL を XML スキーマに変換する際に役立ちます。このオプションは SBCS 文字配列でのみ使用可能です。配列が xsd:string フィールドではなく、固定長の xsd:base64Binary フィールドにマップされるようになります。

マッピング・レベル 1.2 以上の場合

マッピング・レベル 1.2 は、CICS TS 3.1 以降の領域と互換性があります。

以下に示すバッチ・ツールの追加パラメーターを使用して、実行時の文字データおよびバイナリー・データの変換方法をさらに制御できます。

- **CHAR-VARYING**
- **CHAR-VARYING-LIMIT**
- **CHAR-MULTIPLIER**
- **DEFAULT-CHAR-MAXLENGTH**

DFHSC2LS または DFHWS2LS で **CHAR-MULTIPLIER** パラメーターを使用する場合は、このパラメーターの値が文字データに必要なスペース量の計算に使用された後、以下の規則が適用されます。

- DFHSC2LS および DFHWS2LS は次のマッピングを提供します。
  - 最大長が 32 767 バイトより大きい可変長の文字データ・タイプはコンテナーにマップされます。 下限は **CHAR-VARYING-LIMIT** パラメーターを使用して設定できます。 コンテナーの名前を格納するために、16 バイトのフィールドが言語構造に作成されます。 実行時に、文字データはコンテナーに格納され、コンテナー名は言語構造に入ります。
  - 最大長が 32 768 バイトより小さい可変長の文字データ・タイプは、C/C++ および Enterprise PL/I を除くすべての言語で VARYING 構造にマップされます。C/C++ ではこれらのデータはヌル終了ストリングにマップされ、

Enterprise PL/I ではこれらのデータ・タイプは VARYINGZ 構造にマップされます。 **CHAR-VARYING** パラメーターを使用して、可変長の文字データがマップされる方法を選択できます。

– 最大長が 32 768 バイト未満の可変長バイナリー・データ型は、すべての言語の VARYING 構造にマップされます。最大長が 32 768 バイト以上である場合、データはコンテナーにマップされます。 コンテナーの名前を格納するために、16 バイトのフィールドが言語構造に作成されます。 実行時に、バイナリー・データはコンテナーに格納され、コンテナー名は言語構造に入ります。

これらに関連付けられた長さを持たない文字データ型が XML スキーマに存在する場合は、DFHWS2LS または DFHSC2LS の **DEFAULT-CHAR-MAXLENGTH** パラメーターを使用して、デフォルトの長さを割り当てることができます。

DFHLS2SC および DFHLS2WS は次のマッピングを提供します。

- 文字フィールドは `xsd:string` データ・タイプにマップされ、実行時に固定長フィールドまたはヌル終了ストリングとして処理できます。 PL/I を除くすべての言語での、実行時の可変長文字データの処理方法を、**CHAR-VARYING** パラメーターを使用して選択できます。
- Base64Binary データ・タイプはデータの最大長が 32 767 バイトより大きいか、長さが定義されていない場合に、コンテナーにマップされます。 データの長さが 32 767 以下である場合は、base64Binary データ・タイプがすべての言語で VARYING 構造にマップされます。

マッピング・レベル 1.1 以上の場合

マッピング・レベル 1.1 は、CICS TS 3.1 以降の領域と互換性があります。

このマッピング・レベルでは、XML 文字とバイナリー・データ型のマッピングが強化されています (特に、XML スキーマで異なる値で定義された `maxLength` 属性と `minLength` 属性を持つ可変長のデータをマップする場合)。データは次のように処理されます。

- 16 MB より大きい固定長を持つ文字およびバイナリー・データ・タイプが、PL/I を除くすべての言語でコンテナーにマップされます。PL/I では、32 767 バイトより大きい固定長の文字およびバイナリー・データ・タイプがコンテナーにマップされます。 コンテナーの名前を格納するために、16 バイトのフィールドが言語構造に作成されます。 実行時に、固定長データはコンテナーに格納され、コンテナー名は言語構造に入ります。

  コンテナーの長さは可変的なので、コンテナーにマップされた固定長データと、XML スキーマや Web サービス記述で指定された固定長が一致するように、スペースやヌルが埋め込まれたり切り捨てられたりすることはありません。データ長が重要な場合、データ長を検査するアプリケーションを作成するか、または CICS 領域での妥当性検査をオンにすることができます。 SOAP および XML の両方の妥当性検査は、パフォーマンスに大きな影響を及ぼします。
- XML スキーマ <list> および <union> データ・タイプは文字フィールドにマップされます。

- スキーマ定義の XML 属性は無視されるのではなく、マップされます。最大 255 の属性が各 XML エレメントで許可されます。 詳しくは、XML 属性のサポートを参照してください。
- `xsi:nil` 属性がサポートされます。 詳しくは、XML 属性のサポートを参照してください。

マッピング・レベル 1.1 のみ

マッピング・レベル 1.1 は、CICS TS 3.1 以降の領域と互換性があります。

このマッピング・レベルでは、XML 文字とバイナリー・データ型のマッピングが強化されています (特に、XML スキーマで異なる値で定義された `maxLength` 属性と `minLength` 属性を持つ可変長のデータをマップする場合)。データは次のように処理されます。

- 可変長のバイナリー・データ・タイプがコンテナーにマップされます。 コンテナーの名前を格納するために、16 バイトのフィールドが言語構造に作成されます。 実行時に、バイナリー・データはコンテナーに格納され、コンテナー名は言語構造に入ります。
- 最大長が 32 767 バイトより大きい可変長の文字データ・タイプはコンテナーにマップされます。 コンテナーの名前を格納するために、16 バイトのフィールドが言語構造に作成されます。 実行時に、文字データはコンテナーに格納され、コンテナー名は言語構造に入ります。
- 固定長が 16 MB 未満の文字とバイナリー・データ型は、PL/I 以外のすべての言語の固定長フィールドにマップされます。 PL/I の場合、32 767 バイト以下の固定長の文字とバイナリー・データ型は固定長フィールドにマップされます。
- CICS は、hexBinary 形式のデータのエンコードとデコードは行いますが、base64Binary 形式のデータのエンコードとデコードは行いません。 XML スキーマの Base64Binary データ・タイプは言語構造内のフィールドにマップされます。フィールドのサイズは、式: 4×(ceil(z/3)) を使用して計算されます。各部の意味は次のとおりです。
  - *z* は XML スキーマのデータ・タイプの長さです。
  - ceil(*x*) は *x* 以上の最小の整数です。

  z の長さが 24 566 バイトより大きい場合、結果として生成される言語構造のコンパイルは失敗します。 24 566 バイトより大きい base64Binary データの場合、マッピング・レベル 1.2 を使用する必要があります。マッピング・レベル 1.2 では、言語構造内のフィールドを使用する代わりに、base64Binary データをコンテナーにマップできます。

マッピング・レベル 1.0 のみ

マッピング・レベル 1.0 は、CICS TS 3.1 以降の領域と互換性があります。

以下の制限に注意してください。これらの制限は、1.0 より上のマッピング・レベルでは変更されています。

- DFHSC2LS および DFHWS2LS は XML スキーマの文字およびバイナリー・データ・タイプを、言語構造内の固定長フィールドにマップします。 次の部分的な XML スキーマをご覧ください。

```
<xsd:element name="example">
<xsd:simpleType>
<xsd:restriction base="xsd:string">
<xsd:maxLength value="33000"/>
</xsd:restriction>
</xsd:simpleType>
</xsd:element>
```

この部分的な XML スキーマは、次の例のように COBOL 言語構造で表示されます。

```
15 example PIC X(33000)
```

- CICS は、hexBinary 形式のデータのエンコードとデコードは行いますが、base64Binary 形式のデータのエンコードとデコードは行いません。 DFHSC2LS および DFHWS2LS は、Base64Binary データを固定長文字フィールドにマップします。フィールドの内容は、アプリケーション・プログラムによってエンコードまたはデコードされる必要があります。
- DFHSC2LS および DFHWS2LS は処理中に XML 属性を無視します。
- DFHLS2SC および DFHLS2WS は、言語構造内の文字フィールドおよびバイナリー・フィールドを固定長フィールドとして解釈し、これらのフィールドを、maxLength 属性を持つ XML エレメントにマップします。十分なデータがない場合には、実行時に言語構造内のフィールドがスペースやヌルで埋められます。

高水準言語と **XML** のスキーマ・マッピング**:**

CICS アシスタントを使用して高水準言語構造と XML スキーマまたは WSDL ドキュメントとのマッピングを生成します。さらに、CICS アシスタントは、高水準言語構造から XML スキーマまたは WSDL 文書を (またはその逆を) 生成します。

ユーティリティー・プログラム DFHSC2LS と DFHLS2SC を合わせて CICS XML アシスタントといいます。 ユーティリティー・プログラム DFHWS2LS と DFHLS2WS を合わせて CICS Web サービス・アシスタントといいます。

- DFHLS2SC および DFHLS2WS は高水準言語構造を XML スキーマと WSDL 文書にそれぞれマップします。
- DFHSC2LS および DFHWS2LS は XML スキーマと WSDL 文書を高水準言語構造にマップします。

以下に示すように、2 つのマッピングは対称的ではありません。

- DFHLS2SC または DFHLS2WS を使って言語データ構造を処理した結果として生成される XML スキーマまたは WSDL 文書を、相補的なユーティリティー・プログラム (それぞれ DFHSC2LS または DFHWS2LS) を使って処理する場合、最終的な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。ただし、最終的なデータ構造は、最初のデータ構造と論理的には同等です。
- DFHSC2LS または DFHWS2LS を使って XML スキーマまたは WSDL 文書を処理した結果として生成される言語構造を、相補的なユーティリティー・プログラム (それぞれ DFHLS2SC または DFHLS2WS) を使って処理する場合、最終的な XML スキーマまたは WSDL 文書の中の XML スキーマが最初のものと同じになることは期待できません。

- 場合によっては、DFHLS2SC および DFHLS2WS でサポートされない言語構造が DFHSC2LS および DFHWS2LS によって生成されることがあります。

DFHLS2SC および DFHLS2WS で処理される言語構造は、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。

COBOL から **JSON** スキーマへのマッピング**:**

DFHLS2JS ユーティリティー・プログラムは、COBOL データ構造と JSON スキーマ定義との間のマッピングをサポートしています。

COBOL の名前は、以下の規則に従って JSON の名前に変換されます。

- 重複する名前は、1 つ以上の数字が追加されて固有の名前になります。

  例えば、`year` の 2 つのインスタンスは `year` と `year1` になります。
- ハイフンは、下線に置き換えられます。一連の連続するハイフンは、連続する下線で置き換えられます。

  例えば、`current-user--id` は `current_user__id` になります。
- ハイフンで区切られており、大文字のみを含む名前のセグメントは、小文字に変換されます。

  例えば、`CA-REQUEST-ID` は `ca_request_id` になります。
- 数字で始まる名前には、先頭に下線が追加されます。

  例えば、`9A-REQUEST-ID` は `_9a_request_id` になります。

CICS 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 データ記述エレメントは、DFHLS2JS 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 および 77 のデータ記述項目はサポートされていません。 レベル番号が 88 のデータ記述項目は無視されます。
- データ記述項目における以下の節は、サポートされません。

  REDEFINES

  RENAMES (レベル 66)

  DATE FORMAT
- データ記述項目における以下の節は、無視されます。

  BLANK WHEN ZERO

  JUSTIFIED

  VALUE
- SIGN 文節 SIGN TRAILING はサポートされます。 SIGN 文節 SIGN LEADING は、DFHLS2JS で指定されたマッピング・レベルが 1.2 以上の場合のみサポートされます。
- SIGN TRAILING 文節と SIGN LEADING 文節の両方では、SEPARATE CHARACTER はマッピング・レベル 1.2 以上でサポートされます。
- USAGE 節における以下の句は、サポートされません。

OBJECT REFERENCE

POINTER

FUNCTION-POINTER

PROCEDURE-POINTER

- USAGE 文節の次の句は、マッピング・レベル 1.2 以上でサポートされます。

  COMPUTATIONAL-1

  COMPUTATIONAL-2
- 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、S、および Z です。
- PACKED-DECIMAL データ記述項目でサポートされる PICTURE 文字は、9、S、V、および Z です。
- 編集済みの数値データ記述項目でサポートされている PICTURE 文字は、9 と Z だけです。
- MAPPING-LEVEL パラメーターを 1.2 以上に設定して、CHAR-VARYING パラメーターを NULL に設定すると、文字配列は `string` にマップされ、ヌル終了ストリングとして処理されます。
- MAPPING-LEVEL パラメーターを 1.2 以上に設定して、CHAR-VARYING パラメーターを BINARY に設定すると、文字配列は `string` にマップされ、バイナリー・データとして処理されます。
- MAPPING-LEVEL パラメーターを 1.2 以上に設定して、CHAR-VARYING パラメーターを COLLAPSE に設定すると、ストリングの末尾の空白文字は無視されます。
- OCCURS DEPENDING ON 節は、マッピング・レベル 4.0 以上でサポートされます。複雑な OCCURS DEPENDING ON はサポートされません。つまり、構造の最後のフィールドにのみ OCCURS DEPENDING ON がサポートされます。
- OCCURS INDEXED BY 節は、すべてのマッピング・レベルでサポートされます。

COBOL のデータ記述	**JSON** スキーマ定義
PIC X(*n*) PIC A(*n*) PIC G(*n*) DISPLAY-1 PIC N(*n*)	"type":"string", "maxLength": *n*

COBOL のデータ記述	**JSON** スキーマ定義
PIC S9 DISPLAY PIC S99 DISPLAY PIC S999 DISPLAY PIC S9999 DISPLAY PIC S9(*n*) DISPLAY PIC S9(*n*) COMP PIC S9(*n*) COMP-4 PIC S9(*n*) COMP-5 PIC S9(*n*) BINARY	"type":"integer", "minimum":- (*n* + 1), "maximum": *n* 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 DISPLAY PIC 99 DISPLAY PIC 999 DISPLAY PIC 9999 DISPLAY PIC 9(*n*) DISPLAY PIC 9(*n*) COMP PIC 9(*n*) COMP-4 PIC 9(*n*) COMP-5 PIC 9(*n*) BINARY	"type":"integer", "minimum":0, "maximum": *n* 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(*m*)V9(*n*) COMP-3	"type":"number", "format":"decimal", "minimum": *x* , "maximum": *y* , "multipleOf": *z* 各部の意味は次のとおりです。 *x* は、文字「9」のパターンで表現できる最小値です。 *y* は、文字「9」のパターンで表現できる最大値です。 *z* は、使用可能な最小単位 = 1 / 10^n です。

COBOL のデータ記述	**JSON** スキーマ定義
PIC 9(*m*)V9(*n*) COMP-3	"type":"number", "format":"decimal", "minimum": 0, "maximum": *y* , "multipleOf": *z* 各部の意味は次のとおりです。 *y* は、文字「9」のパターンで表現できる最大値です。 *z* は、使用可能な最小単位 = 1 / 10^n です。
PIC S9(*15*) COMP-3 DATETIME=PACKED15 の場合マッピング・レベル 3.0 でサポートされる	"type":"string", "format":"date-time" タイム・スタンプのフォーマットは、RFC3339 により定義されます。
PIC S9(*m*)V9(*n*) DISPLAY マッピング・レベル 1.2 以上でサポートされる	"type":"number", "format":"decimal", "minimum": *x* , "maximum": *y* , "multipleOf": *z* 各部の意味は次のとおりです。 *x* は、文字「9」のパターンで表現できる最小値です。 *y* は、文字「9」のパターンで表現できる最大値です。 *z* は、使用可能な最小単位 = 1 / 10^n です。
COMP-1 マッピング・レベル 1.2 以上でサポートされる **注:** IBM Hexadecimal Floating Point (HFP) データ表記は、JSON に使用される IEEE-754-1985 表記とまったく同じではありません。一部の値が 1 つ表記からもう 1 つの表記に正確に変換されない可能性があります。 極端に大きいまたは小さい値は、float データ型に対して有効ではない場合があります。 HFP 表記との間の変換時に、一部の値で精度が失われる可能性があります。正確な変換が重要になる場合、COMP-1 データ型を使用せずに、固定精度のデータ型を代わりに使用することを検討してください。	"type":"number", "format":"float"
COMP-2 マッピング・レベル 1.2 以上でサポートされる **注:** IBM Hexadecimal Floating Point (HFP) データ表記は、JSON に使用される IEEE-754-1985 表記とまったく同じではありません。一部の値が 1 つ表記からもう 1 つの表記に正確に変換されない可能性があります。 極端に大きいまたは小さい値は、double データ型に対して有効ではない場合があります。 HFP 表記との間の変換時に、一部の値で精度が失われる可能性があります。正確な変換が重要になる場合、COMP-2 データ型を使用せずに、固定精度のデータ型を代わりに使用することを検討してください。	"type":"number", "format":"double"

COBOL のデータ記述	**JSON** スキーマ定義

data description OCCURS *n* TIMES	マッピング・レベル **4.0** 以下
	プリミティブの場合:
	"type":"array"
	"maxItems":
	n
	"minItems":
	n
	"items":{
	"type":"object",
	"properties":{
	name
	:{
	data description JSON
	}
	}
	"required":[
	name
	]
	}
	構造の場合:
	"type":"array"
	"maxItems":
	n
	"minItems":
	n
	"items":{
	data description JSON
	}
	ここで *data description JSON* は、COBOL *data description* の JSON スキーマ表記です。 *name* は、COBOL *data description* の名前です。
	マッピング・レベル **4.1** 以上
	TRUNCATE-NULL-ARRAYS = *DISABLED*
	構造化配列およびプリミティブ配列はどちらも以下のようにマップされます。
	"type":"array"
	"maxItems":
	n
	"minItems":
	n
	"items":{
	data description JSON
	}
	TRUNCATE-NULL-ARRAYS = *ENABLED*
	プリミティブ配列は上記のようにマップされ、構造化配列は以下のようにマップされます。
	"type":"array"
	"maxItems":
	n
	"minItems":
	0
	"items":{
	data description JSON
	}

data description
OCCURS
n
TO
m
TIMES
DEPENDING ON
t
マッピング・レベル 4.0 でサポート
	"field-name"
	:{
	"type":
	"array"
	,
	"maxItems":
	m
	"minItems":
	n
	"items":{
	...
	}
	}
	配列項目の内容は、使用されるデータ型によって異なります。

PIC X OCCURS
n
TIMES
PIC A OCCURS
n
TIMES
PIC G DISPLAY-1 OCCURS
n
TIMES
PIC N OCCURS
n
TIMES
	CHAR-OCCURS =STRING の場合:
	"field-name"
	:{
	"type":
	"string"
	,
	"maxLength":
	n
	}
	これはストリングです。

COBOL のデータ記述	**JSON** スキーマ定義

	CHAR-OCCURS =ARRAY の場合:
	
	マッピング・レベル **4.0** 以下
	
	"field-name"
	:{
	"maxItems":
	m
	,
	"minItems":
	n
	,
	"items":{
	"type":
	"object"
	,
	"properties":{
	"field-name":{
	"type":
	"string"
	,
	"maxLength":1
	}
	},
	"required":[
	"field-name"
	]
	}
	}
	
	これは単一文字の配列です。
	
	マッピング・レベル **4.1** 以上
	
	TRUNCATE-NULL-ARRAYS = *DISABLED*
	
	構造化配列およびプリミティブ配列はどちらも以下のようにマップされます。
	
	"type":"array"
	"maxItems":
	n
	"minItems":
	n
	"items":{
	data description JSON
	}
	
	TRUNCATE-NULL-ARRAYS = *ENABLED*
	
	プリミティブ配列は上記のようにマップされ、構造化配列は以下のようにマップされます。
	
	"type":"array"
	"maxItems":
	n
	"minItems":
	0
	"items":{
	data description JSON
	}

PIC X OCCURS
n
TO
m
TIMES
DEPENDING ON
 	
t
PIC A OCCURS
n
TO
m
TIMES
DEPENDING ON
 	
t
PIC G DISPLAY-1 OCCURS
n
TO
m
TIMES
DEPENDING ON
 	
t
PIC N OCCURS
n
TO
m
TIMES
DEPENDING ON
 	
t
	CHAR-OCCURS =STRING の場合:
	
	"field-name"
	:{
	"type":
	"string"
	,
	"maxLength":
	m
	"minLength":
	n
	}

COBOL のデータ記述	JSON スキーマ定義
PIC N(*n*) USAGE NATIONAL **CHAR-USAGE** =NATIONAL の場合: PIC N(*n*)	マッピング・レベル 4.0 以上の場合: "type":"string", "maxLength": *n* 実行時に、CICS はアプリケーション・データ構造フィールドに UTF-16 データを取り込みます。

JSON スキーマから **COBOL** へのマッピング**:**

DFHJS2LS ユーティリティー・プログラムは、JSON スキーマと COBOL データ構造との間のマッピングをサポートしています。

CICS 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 変数の固有かつ有効なフィールド名を生成します。

1. COBOL 予約語には接頭部「X」が付きます。

   例えば、 DISPLAY は XDISPLAY となります。

2. A から Z、a から z、0 から 9、またはハイフン以外の文字は、「X」で置き換えられます。

   例えば、monthly_total は monthlyXtotal になります。

3. 最後の文字がハイフンである場合は、「X」で置き換えられます。

   例えば、ca-request は ca-requestX となります。

4. 同じスコープ内の重複した名前は、名前の 2 番目以降のインスタンスに 1 つまたは 2 つの数字を追加することによって固有にします。

   例えば、 year の 3 つのインスタンスは year 、 year1 、および year2 になります。

   もし上記の動作が望ましくないなら、ユーザーはユーティリティーの入力として MAPPING-OVERRIDES=NO-ARRAY-NAME-INDEXING を指定できます。これにより、名前の 2 番目以降のインスタンスへの 1 つまたは 2 つの数字の追加が無効になります。

5. JSON スキーマは、"type" 値が "array" で、キーワード "minItems" と "maxItems" が省略されるか、それぞれ別の値が指定されている場合には、変数の基数が可変になるように指定します。変数の基数が可変になるようスキーマで指定されている場合、フィールド名は接尾部 "_cont" および "_num" が付けられて作成されます。

   詳しくは、 324 ページの『DFHJS2LS 内のエレメントの可変配列』を参照してください。

6. JSON スキーマは、"required" キーワード配列 (周りを囲む JSON スキーマ "object" タイプに関連付けられている) に指定されていない変数がオプションとなるように指定します。オプション・フィールドの場合、追加のフィールドは、エレメント名に接尾部 _num が追加されて生成されます。実行時にこれは、JSON データに値が存在しなかったことを示す場合はゼロ、JSON データに値が存在した場合は非ゼロになります。

7. フィールド名は 28 文字に限定されます。接頭部および接尾部を含めて生成された名前がこの長さを超えると、エレメント名が切り捨てられます。

DFHJS2LS は、指定されたマッピング・レベルを使用して、スキーマ・タイプを次の表に従って、 COBOL のデータ記述エレメントにマップします。次の点に注意してください。

- **MAPPING-LEVEL** パラメーターを 1.2 以上に設定して、**CHAR-VARYING** パラメーターを NULL に設定すると、可変長文字データはヌル終了ストリングにマップされ、ヌル終止符として追加された 1 つの文字が割り振られます。
- **MAPPING-LEVEL** パラメーターを 1.2 以上に設定して、**CHAR-VARYING** パラメーターを YES に設定すると、可変長文字データは 2 つの関連エレメント (長さフィールドとデータ・フィールド) にマップされます。以下に例を示します。

```
"textString": {
"type":"string",
"maxLength":10000,
"minLength":1
}
```

これは、次にマップします。

```
15 textString-length PIC S9999 COMP-5 SYNC
15 textString PIC X(10000)
```

JSON スキーマ・キーワード	**COBOL** のデータ記述

以下のすべて:
 	
`"type":"null"`
`"type": []`
	サポートされていない

`"$schema":"http://json-schema.org/draft-04/schema#"`	このキーワードは無視されますが、ドラフト 04 JSON スキーマ仕様と互換性があることが想定されています。
`"title": "same text"` `"description": "more text"`	これらのキーワードは無視されます。
`"format": "<predefined values>"`	"format" キーワードは、生成された構造またはランタイム値のいずれかを変更するために使用されます。サポートされる "format" の使用方法については、この表の後半にある情報を参照してください。

`"type":`
`"array",`
`"items": {<JSON Sub-schema>},`
`"additionalItems": false,`
`"maxItems":`
m
`,`
`"minItems":`
n
	多次元配列は、マッピング・レベル 4.3 以上でサポートされ、混合タイプ配列はサポートされません。
	
	"additionalItems" は、false になることが想定されています。他にサポートされている値はありません。
	
	"minItems" と "maxItems" の両方が存在し、それらが等しい場合、配列は固定基数として処理されます。それ以外の場合、配列は可変基数として処理されます。 324 ページの『DFHJS2LS 内のエレメントの可変配列』を参照してください。

`"type": "array",` `"uniqueItems": true`	"uniqueItems" は、JSON 配列でサポートされていません。

`"type": "object",`
`"additionalProperties": false,`
`"properties": { [ "<element name>": {<JSON Sub-schema>} [,] ]* }`
`"required": [ [ "<element name>" [,] ]* ]`
	現在サポートされている JSON オブジェクトの唯一の形式は、名前付きエレメントの固定セットです。
	
	これは、エレメント名を使用する構造 (または副構造) を生成します。
	
	"additionalProperties" は、値が **ADDITIONAL-PROPERTIES-DEFAULT** パラメーターで設定されていない場合、false と想定されます。
	
	"required" 配列にないか、"required" 配列が存在しない場合、"properties" オブジェクト内のエレメントはすべて "optional" と見なされます。 "optional" エレメントにはゼロから X までの可変オーディナリティーが提供されます (X は 1 または配列の項目の最大数、その項目は配列として定義)。 324 ページの『DFHJS2LS 内のエレメントの可変配列』を参照してください。

`"type": "object",` `"additionalProperties": true`	オブジェクトは前の例のようにマップされ、追加プロパティーをサポートする追加フィールドがあります。"additionalProperties" プロパティーは、**ADDITIONAL-PROPERTIES-DEFAULT** パラメーターで設定されていない場合、false と想定されます。 有効な場合、**ADDITIONAL-PROPERTIES-MAX** パラメーターで指定された値までスペースが割り当てられます。各スペースの文字数は、**ADDITIONAL-PROPERTIES-SIZE** パラメーターで設定されています。個々のプロパティーは、PIC X(*z*) フィールドにマップされます。z は、**ADDITIONAL-PROPERTIES-SIZE** パラメーターによって定義されます。**ADDITIONAL-PROPERTIES-MAX** の値が 0 より大きい場合、プロパティーは、**maxItems** が設定された配列のプロパティーと同じようにマップされます。 **注:** CICS での z/OS Connect の使用など、CICS での JSON サポートを構成する方法はいくつかあります。古い CICS Java パイプライン・テクノロジーを使用する場合、**追加プロパティー**は、com.ibm.cics.json.enableAxis2Handlers JVM システム・プロパティーが true に設定されていないときにのみサポートされます。

`"type": "object",`
`"maxProperties": m,`
`"minProperties": n,`
`"patternProperties": {},`
`"dependencies":`
	これらのキーワードは、いずれも JSON オブジェクトではサポートされません。

JSON スキーマ・キーワード	**COBOL** のデータ記述
"type":"string" "maxLength": *m* "pattern": "regular expression>", "minLength": *l*	PIC X(*z*) ここで *z* の値は *m* に基づいていますが、 *CHAR-VARYING* パラメーターの設定に依存します。 *m* は "maxLength" キーワードに基づき、固定長ストリングとして扱われます。 "pattern" および "minLength" の制約事項は、コメントとしてのみ言語構造に渡されます。
"type":"string" "maxLength": *m*	マッピング・レベル 4.0 以上で CCSID=1200 の場合: PIC N(*z*) USAGE NATIONAL ここで *z* の値は *m* に基づいていますが、 **CHAR-VARYING** パラメーターの設定に依存します。 *m* は "maxLength" キーワードに基づき、固定長ストリングとして扱われます。

"*name*":{
"type":"string",
"format":"date-time"
}
	PIC S9(15) COMP-3
	
	DATETIME=PACKED15 の場合、すべてサポートされます。
	
	"maxLength" および "minLength" は、この形式ではサポートされないことに注意してください。

"*name*":{
"type":"string",
"format":"uri"
}
	PIC X(*m*)
	
	ここで、*m* は "maxLength" キーワードに基づき、固定長ストリングとして扱われます。
	
	マッピング・レベル 4.0 以上で **CCSID=1200** の場合:
	
	PIC N(m) USAGE NATIONAL
	
	ここで、*m* は "maxLength" キーワードに基づき、固定長ストリングとして扱われます。

"*name*":{
"type":"string",
"format":"base64Binary"
}
	PIC X(*m*)
	
	ここで、 *m* は "maxLength" キーワードに基づきます。

"*name*":{
"type":"string",
"format":"hexBinary"
}
	PIC X(*m*)
	
	ここで、 *m* は "maxLength" キーワードに基づきます。

"*name*":{
"type":"string",
"format":"<predefined>"
}
	PIC X(*m*)
	
	ここで、 *m* は "maxLength" キーワードに基づき固定長ストリングとして扱われます。 また、 <predefined> は *email*、 *hostname*、 *ipv4*、 または *ipv6* のいずれかです。関連する "pattern" が使用され、コメントに渡されます。
	
	マッピング・レベル 4.0 以上で **CCSID=1200** の場合:
	
	PIC N(*m*) USAGE NATIONAL
	
	ここで、 *m* は "maxLength" キーワードに基づき固定長ストリングとして扱われます。 また、 <predefined> は *email*、 *hostname*、 *ipv4*、 または *ipv6* のいずれかです。関連する "pattern" が使用され、コメントに渡されます。

"type":"boolean"	PIC X DISPLAY 値 x'00' は false、x'01' は true を意味します。
"type": "integer", "exclusiveMaximum": true, "exclusiveMinimum": true, "multipleOf": n	"exclusiveMaximum" および "exclusiveMinimum" の制約事項は、コメントとしてのみ言語構造に渡されます。 "multipleOf" は無視されます。
"type"="integer", minimum=0, maximum=255	マッピング・レベル 3.0 以下: PIC X DISPLAY マッピング・レベル 4.0 以上 (または、マッピング・レベルとは関係なく、INTEGER-AS-PIC9 パラメーターが指定されている場合): PIC 9(*z*) COMP-5 SYNC または PIC 9(*z*) DISPLAY ここで $10^{(z-1)} < m <= 10^{z}$

JSON スキーマ・キーワード	**COBOL** のデータ記述
"type":"integer", minimum:-128, maximum:127	マッピング・レベル 3.0 以下: PIC X DISPLAY マッピング・レベル 4.0 以上 (または、マッピング・レベルとは関係なく、INTEGER-AS-PIC9 パラメーターが指定されている場合): PIC S9(*z*) COMP-5 SYNC または PIC S9(*z*) DISPLAY ここで $10^{(z-1)} < m <= 10^{z}$
"type":"integer", minimum:0, maximum; *m*	PIC 9(*z*) COMP-5 SYNC または PIC 9(*z*) DISPLAY ここで $10^{(z-1)} < m <= 10^{z}$
"type":"integer", minimum:- *m* , maximum: *m* -1	PIC S9(*z*) COMP-5 SYNC または PIC S9(*z*) DISPLAY ここで $10^{(z-1)} < m <= 10^{z}$
"type": "number", "maximum": m, "minimum": n, "exclusiveMaximum": true, "exclusiveMinimum": true, "multipleOf": n	"maximum"、"minimum"、"exclusiveMaximum" および "exclusiveMinimum" の制約事項は、コメントとしてのみ言語構造に渡されます。 "multipleOf" は無視されます。
"type":"number" "format":"decimal"	PIC 9(*p*)V9(*n*) COMP-3 ここで、*p* および *n*はデフォルト値です。
"type":"number" "format":"float"	マッピング・レベル 1.1 以下: • PIC X(32) マッピング・レベル 1.2 以上の場合 • COMP-1 **注:** IBM Hexadecimal Floating Point (HFP) データ表記は、JSON に使用される IEEE-754-1985 表記とまったく同じではありません。一部の値が 1 つ表記からもう 1 つの表記に正確に変換されない可能性があります。 極端に大きいまたは小さい値は、float データ型に対して有効ではない場合があります。 HFP 表記との間の変換時に、一部の値で精度が失われる可能性があります。正確な変換が重要になる場合、COMP-1 データ型を使用せずに、固定精度のデータ型を代わりに使用することを検討してください。
"type":"number" "format":"double"	マッピング・レベル 1.1 以下: • PIC X(32) マッピング・レベル 1.2 以上の場合 • COMP-2 **注:** IBM Hexadecimal Floating Point (HFP) データ表記は、JSON に使用される IEEE-754-1985 表記とまったく同じではありません。一部の値が 1 つ表記からもう 1 つの表記に正確に変換されない可能性があります。 極端に大きいまたは小さい値は、double データ型に対して有効ではない場合があります。 HFP 表記との間の変換時に、一部の値で精度が失われる可能性があります。正確な変換が重要になる場合、COMP-2 データ型を使用せずに、固定精度のデータ型を代わりに使用することを検討してください。

JSON スキーマ・キーワード	COBOL のデータ記述

`oneOf: [ { "properties": { "A": {...} } } ,`
`{ "properties": { "B": {...} } } ]`
`anyOf: [ { "properties": { "A": {...} } } ,`
`{ "properties": { "B": {...} } } ]`
`allOf: [ { "properties": { "A": {...} } } ,`
`{ "properties": { "B": {...} } } ]`
	オプションの配列を使用した論理パスは、{ "properties": { "A":..., "B":... } } というタイプの単一オブジェクトが定義されている場合と同様にマージされます。JSON プロパティーはそれぞれオプションとして扱われます。同じプロパティー名の競合する定義がサブオプションに含まれている場合は、エラー・メッセージが発行されます。例えば、単一プロパティーがストリング (パス A 内) とオブジェクト (パス B 内) の両方として定義されている場合、そのような定義はサポートされず、エラー・メッセージが表示されます。
	
	JSON スキーマで複雑な論理構造を定義できますが、複雑な論理構造内で暗黙的に指定された細かい点は、言語構造へのマッピングで失われる可能性があります。変換プロセスは、スキーマからの組み合わせルールを強制しようとしません。指定された JSON プロパティーが存在するか存在しないかを示す言語構造フィールドとのみ対話します。
	
	サブオプションに同じプロパティー名で互換性のある定義が含まれている場合、DFHJS2LS は、制約の関連パターンをマージしようとしますが、プロセスではいくつかの細かい点が失われる場合があります。例えば、以下の定義を想定します。
	
	`"A": { "oneOf": [ { "type": "string", "maxLength": 5}, {"type": "string", "minLength": 7, "maxLength": 8} ] }`
	
	『A』 は、最大 5 文字の長さのストリング、または長さが 7 から 8 文字のストリングのいずれかとして定義されています。構成のマージにより、6 文字の長さが無効であることは認識されず、0 から 8 文字のストリングにマッピングされる可能性があります。
	
	論理構成を含むほとんどのシナリオでは単純な言語構造にマップされますが、複雑な論理構成では、マッピング処理中に妥協が図られることがあります。最良の結果を得るために、JSON スキーマで論理構成を使用せずに、同じ JSON プロパティーに代替宣言を定義してください。

注: CICS では、signed long の最大値 (2^63 - 1) より大きい整数値を変換できません。ただし、整数値が引用符で囲まれている場合は変換できます。

注: 数値型のスキーマに指定された最小値および最大値は、COBOL データ型にマップされる目的でのみ使用されます。実行時にデータがこれらの値に基づいて検証されることはありません。

表に示したスキーマ・タイプの一部は、`minimum` キーワードおよび `maximum` キーワードに指定された値 (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。

  ```
  "maximum":
  a
  "minimum":
  -a
  ```

  ここで、*a* は数字 9 からなるストリングです。

- 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。

  ```
  "maximum":
  a
  "minimum":0
  ```

  ここで、*a* は数字 9 からなるストリングです。

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++** から **JSON** スキーマへのマッピング:

DFHLS2JS ユーティリティー・プログラムは、C および C++ データ・タイプと JSON スキーマ定義との間のマッピングをサポートしています。

C および C++ の名前は、次の規則に従って JSON の名前に変換されます。

1. JSON プロパティー名に含まれる無効な文字は、「X」で置き換えられます。

   例えば、 monthly-total は monthlyXtotal となります。
2. 重複する名前は、1 つ以上の数字が追加されて固有の名前になります。

   例えば、year の 2 つのインスタンスは year と year1 になります。

DFHLS2JS 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++ のタイプは DFHLS2JS ではサポートされません。 _Packed 修飾子が構造用にサポートされています。 以下の制限が適用されます。

- ヘッダー・ファイルには、最上位の struct インスタンスが含まれていなければなりません。
- 自身をメンバーとして含む構造化タイプを宣言することはできません。
- 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
  long double
  wchar_t (C++ のみ)
- 以下は、ヘッダー・ファイル内に存在する場合、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)
  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、以下の項目を指定できません。
  共用体
  クラス宣言
  列挙型データ・タイプ
  ポインター型変数
  テンプレート宣言
  定義済みマクロ (名前の先頭と末尾が 2 つの下線文字 (__) 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、32 ビットのコンパイラーを使用して int が 4 バイトに確実にマップされるようにする必要があります。
- 次の C++ 予約済みキーワードはサポートされません。

  explicit

  using

  namespace

  typename

  typeid
- **MAPPING-LEVEL** パラメーターを 1.2 以上に設定して、**CHAR-VARYING** パラメーターを NULL に設定すると、文字配列は string にマップされ、ヌル終了ストリングとして処理されます。
- **MAPPING-LEVEL** パラメーターを 1.2 以上に設定して、**CHAR-VARYING** パラメーターを BINARY に設定すると、文字配列は xsd:base64Binary にマップされ、バイナリー・データとして処理されます。
- **MAPPING-LEVEL** パラメーターを 1.2 以上に設定して、**CHAR-VARYING** パラメーターを COLLAPSE に設定すると、<xsd:whiteSpace value="collapse"/> がストリング用に生成されます。

C および **C++** のデータ・タイプ	スキーマの **simpleType**
char[*z*]	"type":"string" "maxlength": *z*
char16_t[*n*]	マッピング・レベル 4.0 以上の場合: "type":"string" "maxlength": *n* 実行時に、CICS はアプリケーション・データ構造フィールドに UTF-16 データを取り込みます。
char[8] DATETIME=PACKED15 の場合マッピング・レベル 3.0 以上でサポートされる	"type":"string" "format":"date-time" タイム・スタンプのフォーマットは、RFC3339 により定義されます。
char short int long long long	"type":"integer", "minimum":- (*n* + 1), "maximum": *n* ここで、*n* はプリミティブで表すことができる最大値です。
unsigned char unsigned short unsigned int unsigned long unsigned long long	"type":"integer", "minimum":0, "maximum": *n* ここで、*n* はプリミティブで表すことができる最大値です。

C および **C++** のデータ・タイプ	スキーマの **simpleType**
`bool` (C++ のみ)	`"type":"boolean"`

`float`
 	
マッピング・レベル 1.2 以上でサポートされる
	`"type":"number",`
	`"format":"float"`
	
	注: IBM Hexadecimal Floating Point (HFP) データ表記は、JSON に使用される IEEE-754-1985 表記とまったく同じではありません。一部の値が、ある表記から別の表記に正確に変換されない可能性があります。極端に大きいまたは小さい値は、float データ型に対して有効ではない場合があります。HFP 表記との間の変換時に、一部の値で精度が失われる可能性があります。正確な変換が重要になる場合、float データ型を使用せずに、固定精度のデータ型を代わりに使用することを検討してください。

`double`
 	
マッピング・レベル 1.2 以上でサポートされる
	`"type":"number",`
	`"format":"double"`
	
	注: IBM Hexadecimal Floating Point (HFP) データ表記は、JSON に使用される IEEE-754-1985 表記とまったく同じではありません。一部の値が、ある表記から別の表記に正確に変換されない可能性があります。極端に大きいまたは小さい値は、double データ型に対して有効ではない場合があります。HFP 表記との間の変換時に、一部の値で精度が失われる可能性があります。正確な変換が重要になる場合、double データ型を使用せずに、固定精度のデータ型を代わりに使用することを検討してください。

type name[*n*]	プリミティブの場合:
	
	`"type":"array"`
	`"maxItems":`
	n
	`"minItems":`
	n
	`"items":{`
	`"type":"object",`
	`"properties":{`
	name
	`:{`
	type JSON
	`}`
	`}`
	`"required":[`
	name
	`]`
	`}`
	
	構造体の場合:
	
	`"type":"array"`
	`"maxItems":`
	n
	`"minItems":`
	n
	`"items":{`
	type JSON
	`}`
	
	ここで *type JSON* は、C または C++ 型の JSON スキーマ表記です。

JSON スキーマから **C** および **C++** へのマッピング**:**

DFHJS2LS ユーティリティー・プログラムは、JSON スキーマと、C および C++ データ・タイプとの間のマッピングをサポートしています。

CICS 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 変数および C++ 変数の固有かつ有効なフィールド名を生成します。

1. A から Z、a から z、0 から 9、または _ 以外の文字は、「X」で置き換えられます。

   例えば、monthly-total は monthlyXtotal になります。
2. 先頭文字が英字ではない場合は、先頭文字が「X」で置き換えられます。

   例えば、_monthlysummary は Xmonthlysummary になります。
3. 同じスコープ内の重複した名前は、名前の 2 番目以降のインスタンスに 1 つまたは 2 つの数字を追加することによって固有にします。

   例えば、 year の 3 つのインスタンスは year 、 year1 、および year2 になります。
4. JSON スキーマは、"type" 値が "array" で、キーワード "minItems" と "maxItems" が省略されるか、それぞれ別の値が指定されている場合には、変数の基数が可変になるように指定します。変数の基数が可変になるようスキーマで指定されている場合、フィールド名は接尾部 "_cont" および "_num" が付けられて作成されます。

   詳しくは、 324 ページの『DFHJS2LS 内のエレメントの可変配列』を参照してください。
5. JSON スキーマは、"required" キーワード配列 (周りを囲む JSON スキーマ "object" タイプに関連付けられている) に指定されていない変数がオプションとなるように指定します。 オプション・フィールドの場合、追加のフィールドは、エレメント名に接尾部 _num が追加されて生成されます。実行時にこれは、JSON データに値が存在しなかったことを示す場合はゼロ、JSON データに値が存在した場合は非ゼロになります。
6. フィールド名は 50 文字までに制限されています。接頭部および接尾部を含めて生成された名前がこの長さを超えると、エレメント名が切り捨てられます。

DFHJS2LS は、次の表に従って JSON スキーマ・タイプ値を C および C++ データ・タイプにマップします。次の規則も適用されます。

- **MAPPING-LEVEL** パラメーターを 1.2 以上に設定して、**CHAR-VARYING** パラメーターを NULL に設定すると、可変長文字データはヌル終了ストリングにマップされ、ヌル終止符として追加された 1 つの文字が割り振られます。
- **MAPPING-LEVEL** パラメーターを 1.2 以上に設定して、**CHAR-VARYING** パラメーターを YES に設定すると、可変長文字データは 2 つの関連エレメント (長さフィールドとデータ・フィールド) にマップされます。

JSON スキーマ・キーワード	**C** および **C++** のデータ・タイプ

以下のすべて:
`"type":"null"`
`"type": []`
	サポートされていない

JSON スキーマ・キーワード	**C** および **C++** のデータ・タイプ
`"$schema":"http://json-schema.org/draft-04/schema#"`	このキーワードは無視されますが、ドラフト 04 JSON スキーマ仕様と互換性があることが想定されています。
`"title": "same text"` `"description": "more text"`	これらのキーワードは無視されます。
`"format": "<predefined values>"`	"format" キーワードは、生成された構造またはランタイム値のいずれかを変更するために使用されます。サポートされる "format" の使用方法については、以下を参照してください。

`"type":`
`"array",`
`"items": {<JSON Sub-schema>},`
`"additionalItems": false,`
`"maxItems":`
m
`,`
`"minItems":`
n
	多次元配列は、マッピング・レベル 4.3 以上でサポートされ、混合タイプ配列はサポートされません。
	
	"additionalItems" は、false になることが想定されています。他にサポートされている値はありません。
	
	"minItems" と "maxItems" の両方が存在し、それらが等しい場合、配列は固定基数として処理されます。それ以外の場合、配列は可変基数として処理されます。 324 ページの『DFHJS2LS 内のエレメントの可変配列』を参照してください。

`"type": "array",` `"uniqueItems": true`	"uniqueItems" は、JSON 配列でサポートされていません。 <JSON Sub-schema> はサポートされている "type" を定義する必要がありますが、"type" を "array" にすることはできません。 これは、生成される言語構造に対する制約事項です。

`"type": "object",`
`"additionalProperties": false,`
`"properties": { [ "<element name>": {<JSON Sub-schema>} [,] ]* }`
`"required": [ [ "<element name>" [,] ]* ]`
	現在サポートされている JSON オブジェクトの唯一の形式は、名前付きエレメントの固定セットです。
	
	これは、エレメント名を使用して構造 (または副構造) を生成します。
	
	"additionalProperties" は、値が **ADDITIONAL-PROPERTIES-DEFAULT** パラメーターで設定されていない場合、false と想定されます。
	
	"required" 配列にないか、"required" 配列が存在しない場合、"properties" オブジェクト内のエレメントはすべて "optional" と見なされます。 "optional" エレメントにはゼロから X までの可変オーディナリティーが提供されます (X は 1 または配列の項目の最大数、その項目は配列として定義)。 324 ページの『DFHJS2LS 内のエレメントの可変配列』を参照してください。

`"type": "object",` `"additionalProperties": true`	オブジェクトは前の例のようにマップされ、追加プロパティーをサポートする追加フィールドがあります。"additionalProperties" プロパティーは、**ADDITIONAL-PROPERTIES-DEFAULT** パラメーターで設定されていない場合、false と想定されます。 有効な場合、**ADDITIONAL-PROPERTIES-MAX** パラメーターで指定された値までスペースが割り当てられます。各スペースの文字数は、**ADDITIONAL-PROPERTIES-SIZE** パラメーターで設定されています。個々のプロパティーは、PIC X(*z*) フィールドにマップされます。*z* は、**ADDITIONAL-PROPERTIES-SIZE** パラメーターによって定義されます。**ADDITIONAL-PROPERTIES-MAX** の値が 0 より大きい場合、プロパティーは、**maxItems** が設定された配列のプロパティーと同じようにマップされます。**注:** CICS での z/OS Connect の使用など、CICS での JSON サポートを構成する方法はいくつかあります。古い CICS Java パイプライン・テクノロジーを使用する場合、**追加プロパティー**は、com.ibm.cics.json.enableAxis2Handlers JVM システム・プロパティーが true に設定されていないときにのみサポートされます。

`"type": "object",`
`"maxProperties": m,`
`"minProperties": n,`
`"patternProperties": {},`
`"dependencies":`
	これらのキーワードは、いずれも JSON オブジェクトではサポートされません。

`"type":"string"`
`"maxLength":`
m
`"pattern": "regular expression>",`
`"minLength":`
l
	char[*z*]
	
	ここで *z* の値は *m* に基づいていますが、 **CHAR-VARYING** パラメーターの設定に依存します。
	
	m は "maxLength" キーワードに基づき、固定長ストリングとして扱われます。
	
	"pattern" および "minLength" の制約事項は、コメントとしてのみ言語構造に渡されます。

`"type":"string"`
`"maxLength":`
m
	マッピング・レベル 4.0 以上で CCSID=1200 の場合:
	
	char16_t[*z*]
	
	ここで *z* の値は *m* に基づいていますが、 **CHAR-VARYING** パラメーターの設定に依存します。
	
	m は "maxLength" キーワードに基づき、固定長ストリングとして扱われます。

JSON スキーマ・キーワード	**C** および **C++** のデータ・タイプ

"*name*":{
"type":"string",
"format":"date-time"
}
	char[*8*]
	
	DATETIME=PACKED15 の場合、すべてサポートされます。

"*name*":{
"type":"string",
"format":"uri"
}
	char[*m*]
	
	ここで、*m* は "maxLength" キーワードに基づき、固定長ストリングとして扱われます。
	
	マッピング・レベル 4.0 以上で **CCSID=1200** の場合:
	
	char16_t[*m*]
	
	ここで、*m* は "maxLength" キーワードに基づき、固定長ストリングとして扱われます。

"*name*":{
"type":"string",
"format":"base64Binary"
}
	char[*m*]
	
	ここで、*m* は "maxLength" キーワードに基づいています。

"*name*":{
"type":"string",
"format":"hexBinary"
}
	char[*m*]
	
	ここで、*m* は "maxLength" キーワードに基づいています。

"*name*":{
"type":"string",
"format":"<predefined>"
}
	char[*m*]
	
	ここで、 *m* は "maxLength" キーワードに基づき固定長ストリングとして扱われます。 また、 <predefined> は *email*、 *hostname*、 *ipv4*、 または *ipv6* のいずれかです。関連する "pattern" が使用され、コメントに渡されます。
	
	マッピング・レベル 4.0 以上で **CCSID=1200** の場合:
	
	char16_t[*m*]
	
	ここで、 *m* は "maxLength" キーワードに基づき固定長ストリングとして扱われます。 また、 <predefined> は *email*、 *hostname*、 *ipv4*、 または *ipv6* のいずれかです。関連する "pattern" が使用され、コメントに渡されます。

"type":"boolean"	bool (C++ のみ) short (C のみ)
"type": "integer", "exclusiveMaximum": true, "exclusiveMinimum": true, "multipleOf": n	"exclusiveMaximum" および "exclusiveMinimum" の制約事項は、コメントとしてのみ言語構造に渡されます。 "multipleOf" は無視されます。
"type":"integer", minimum:-128, maximum:127	signed char
"type":"integer", minimum:0, maximum:255	unsigned char
"type":"integer", minimum:-32768, maximum:32767	short
"type":"integer", minimum:0, maximum:65535	unsigned short
"type":"integer", minimum:-2147483648, maximum:2147483647	int
"type":"integer", minimum:0, maximum:4294967295	unsigned int
"type":"integer", minimum:-9223372036854775808, maximum:9223372036854775807	long long
"type":"integer", minimum:0, maximum:18446744073709551615	unsigned long long
"type": "number", "maximum": m, "minimum": n, "exclusiveMaximum": true, "exclusiveMinimum": true, "multipleOf": n	"maximum"、"minimum"、"exclusiveMaximum" および "exclusiveMinimum" の制約事項は、コメントとしてのみ言語構造に渡されます。 "multipleOf" は無視されます。

JSON スキーマ・キーワード	**C** および **C++** のデータ・タイプ
`"type":"number"` `"format":"float"`	マッピング・レベル 1.1 以下: • `char[32]` マッピング・レベル 1.2 以上の場合 • `float(*)` **注:** IBM Hexadecimal Floating Point (HFP) データ表記は、JSON に使用される IEEE-754-1985 表記とまったく同じではありません。一部の値が、ある表記から別の表記に正確に変換されない可能性があります。極端に大きいまたは小さい値は、float データ型に対して有効ではない場合があります。 HFP 表記との間の変換時に、一部の値で精度が失われる可能性があります。正確な変換が重要になる場合、float データ型を使用せずに、固定精度のデータ型を代わりに使用することを検討してください。
`"type":"number"` `"format":"double"`	マッピング・レベル 1.0 以下: • `char[32]` マッピング・レベル 1.2 以上の場合 • `double(*)` **注:** IBM Hexadecimal Floating Point (HFP) データ表記は、JSON に使用される IEEE-754-1985 表記とまったく同じではありません。一部の値が、ある表記から別の表記に正確に変換されない可能性があります。極端に大きいまたは小さい値は、double データ型に対して有効ではない場合があります。 HFP 表記との間の変換時に、一部の値で精度が失われる可能性があります。正確な変換が重要になる場合、double データ型を使用せずに、固定精度のデータ型を代わりに使用することを検討してください。

`oneOf: [ { "properties": { "A": {...} } } ,`
`{ "properties": { "B": {...} } } ]`
`anyOf: [ { "properties": { "A": {...} } } ,`
`{ "properties": { "B": {...} } } ]`
`allOf: [ { "properties": { "A": {...} } } ,`
`{ "properties": { "B": {...} } } ]`
	オプションの配列を使用した論理パスは、`{ "properties": { "A":..., "B":... } }` というタイプの単一オブジェクトが定義されている場合と同様にマージされます。JSON プロパティーはそれぞれオプションとして扱われます。同じプロパティー名の競合する定義がサブオプションに含まれている場合は、エラー・メッセージが発行されます。例えば、単一プロパティーがストリング (パス A 内) とオブジェクト (パス B 内) の両方として定義されている場合、そのような定義はサポートされず、エラー・メッセージが表示されます。
	JSON スキーマで複雑な論理構造を定義できますが、複雑な論理構造内で暗黙的に指定された細かい点は、言語構造へのマッピングで失われる可能性があります。変換プロセスは、スキーマからの組み合わせルールを強制しようとしません。指定された JSON プロパティーが存在するか存在しないかを示す言語構造フィールドとのみ対話します。
	サブオプションに同じプロパティー名で互換性のある定義が含まれている場合、DFHJS2LS は、制約の関連パターンをマージしようとしますが、プロセスではいくつかの細かい点が失われる場合があります。例えば、以下の定義を想定します。
	`"A": { "oneOf": [ { "type": "string", "maxLength": 5}, {"type": "string", "minLength": 7, "maxLength": 8} ] }`
	『A』 は、最大 5 文字の長さのストリング、または長さが 7 から 8 文字のストリングのいずれかとして定義されています。構成のマージにより、6 文字の長さが無効であることは認識されず、0 から 8 文字のストリングにマッピングされる可能性があります。
	論理構成を含むほとんどのシナリオでは単純な言語構造にマップされますが、複雑な論理構成では、マッピング処理中に妥協が図られることがあります。最良の結果を得るために、JSON スキーマで論理構成を使用せずに、同じ JSON プロパティーに代替宣言を定義してください。

注**:** CICS では、signed long の最大値 (2^63 - 1) より大きい整数値を変換できません。ただし、整数値が引用符で囲まれている場合は変換できます。

注**:** 数値型のスキーマに指定された最小値および最大値は、C または C++ データ型にマップされる目的でのみ使用されます。実行時にデータがこれらの値に基づいて検証されることはありません。

PL/I から **JSON** スキーマへのマッピング**:**

DFHLS2JS ユーティリティー・プログラムは、PL/I データ構造と JSON スキーマ定義との間のマッピングをサポートしています。Enterprise PL/I コンパイラーと古

い PL/I コンパイラーの間には相違点があるため、PLI-ENTERPRISE と PLI-OTHER の 2 つの言語オプションがサポートされます。

PL/I の名前は、次の規則に従って JSON の名前に変換されます。

1. JSON プロパティー名に含まれる無効な文字は、「x」で置き換えられます。

   例えば、monthly$total は monthlyxtotal になります。

2. 重複する名前は、1 つ以上の数字が追加されて固有の名前になります。

   例えば、year の 2 つのインスタンスは year と year1 になります。

DFHLS2JS は次の表に従って PL/I 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 PL/I タイプは DFHLS2JS でサポートされません。 次の制約事項も適用されます。

- COMPLEX 属性を持つデータ項目はサポートされません。
- FLOAT 属性を持つデータ項目は、マッピング・レベル 1.2 以上でサポートされます。 Enterprise PL/I FLOAT IEEE はサポートされません。
- VARYING および VARYINGZ の純粋な DBCS ストリングは、マッピング・レベル 1.2 以上でサポートされます。
- 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。
- PICTURE ストリングはサポートされません。
- ORDINAL データ項目は、FIXED BINARY(7) データ・タイプとして扱われます。
- **MAPPING-LEVEL** パラメーターを 1.2 以上に設定して、**CHAR-VARYING** パラメーターを NULL に設定すると、文字配列は string にマップされ、ヌル終了ストリングとして処理されます。このマッピングは Enterprise PL/I には適用されません。
- **MAPPING-LEVEL** パラメーターを 1.2 以上に設定して、**CHAR-VARYING** パラメーターを BINARY に設定すると、文字配列は string にマップされ、バイナリー・データとして処理されます。
- **MAPPING-LEVEL** パラメーターが 1.2 以上に設定されていて、**CHAR-VARYING** パラメーターが COLLAPSE に設定されていると、先頭と末尾の空白文字が削除され、複数のスペースがシングル・スペースに置換されます。

DFHLS2JS は、PL/I の埋め込みアルゴリズムを完全には実装しないため、データ構造で埋め込みバイトを明示的に宣言する必要があります。 DFHLS2JS は、埋め込みバイトがないことを検出すると、メッセージを発行します。それぞれの最上位構造は、ダブルワード境界で開始して、構造内のそれぞれのバイトは正しい境界にマップされている必要があります。 次のコード・フラグメントについて考えてみます。

```
 3 FIELD1 FIXED BINARY(7),
3 FIELD2 FIXED BINARY(31),
3 FIELD3 FIXED BINARY(63);
```

この例では以下のようになります。
- FIELD1 の長さは 1 バイトで、任意の境界に合わせることができます。
- FIELD2 の長さは 4 バイトで、フルワード境界に合わせる必要があります。
- FIELD3 の長さは 8 バイトで、ダブルワード境界に合わせる必要があります。

Enterprise PL/I コンパイラーはフィールドを以下の順序に調整します。

1. FIELD3 の境界要件が最も厳しいため、FIELD3 が最初に配置されます。
2. FIELD2 は FIELD3 の直前のフルワード境界に合わせます。
3. FIELD1 が FIELD3 の直前のバイト境界に合わせます。

最後に、構造全体がフルワード境界に合うように、コンパイラーは FIELD1 の直前に 3 つの埋め込みバイトを挿入します。

DFHLS2JS は同等の埋め込みバイトを挿入しないため、DFHLS2JS が構造を処理する前にこれらの埋め込みバイトを明示的に宣言する必要があります。 以下に例を示します。

```
 3 PAD1 FIXED BINARY(7),
3 PAD2 FIXED BINARY(7),
3 PAD3 FIXED BINARY(7),
3 FIELD1 FIXED BINARY(7),
3 FIELD2 FIXED BINARY(31),
3 FIELD3 FIXED BINARY(63);
```

あるいは、すべてのフィールドを位置合わせされないと宣言するよう構造を変更して、この構造を使用するアプリケーションを再コンパイルすることもできます。 PL/I 構造上のメモリー位置合わせ要件について詳しくは、Enterprise PL/I for z/OS の製品情報を参照してください。

PL/I のデータ記述	**JSON** スキーマ定義
FIXED BINARY (*n*)	"type":"integer", "minimum":- (*n* + 1), "maximum": *n* ここで、*n* はプリミティブで表すことができる最大値です。
UNSIGNED FIXED BINARY(*n*) **制約事項:** Enterprise PL/I のみ	"type":"integer", "minimum":0, "maximum": *n* ここで、*n* はプリミティブで表すことができる最大値です。

PL/I のデータ記述	**JSON** スキーマ定義
FIXED DECIMAL(*n*,*m*)	"type":"number", "format":"decimal", "multipleOf": *x* , "maximum": *y* , "minimum":- *z* ここで、 *x* は、使用可能な最小単位 = 1 / 10^m です。 *y* は、*n* と *m* の組み合わせで表現できる最大値です。 *z* は、*n* と *m* の組み合わせで表現できる最大値です。
FIXED DECIMAL(15) **DATETIME=PACKED15** の場合マッピング・レベル 3.0 以上でサポートされる	"type":"string", "format":"date-time" タイム・スタンプのフォーマットは、RFC3339 により定義されます。
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。	"type":"string" "maxLength": *m* ここで、 *m* = *n* /8
CHARACTER(*n*) VARYING および VARYINGZ はマッピング・レベル 1.2 以上でもサポートされる **制約事項:** VARYINGZ は Enterprise PL/I でのみサポートされる	"type":"string" "maxLength": *n*
GRAPHIC(*n*) VARYING および VARYINGZ はマッピング・レベル 1.2 以上でもサポートされる **制約事項:** VARYINGZ は Enterprise PL/I でのみサポートされる	マッピング・レベル 1.0 および 1.1 で、 *m* = 2**n* の場合: "type":"string" "maxLength": *m* マッピング・レベル 1.2 以上の場合: "type":"string" "maxLength": *n*

PL/I のデータ記述	**JSON** スキーマ定義

WIDECHAR(*n*)
制約事項: Enterprise PL/I のみ
	マッピング・レベル 1.0 および 1.1 で、 *m* = 2*n* の場合:
	`"type":"string"`
	`"maxLength":`
	m
	
	マッピング・レベル 1.2 以上の場合:
	`"type":"string"`
	`"maxLength":`
	n
	
	マッピング・レベル 4.0 以上では、CICS はアプリケーション・データ構造フィールドに UTF-16 データを取り込みます。
	`<xsd:simpleType>`
	`<xsd:restriction base="xsd:string">`
	`<xsd:maxlength value="n"/>`
	`<xsd:whiteSpace value="preserve"/>`
	`</xsd:restriction>`
	`</xsd:simpleType>`

ORDINAL
制約事項: Enterprise PL/I のみ
	`"type":"integer",`
	`"minimum":0,`
	`"maximum":255`

BINARY FLOAT(*n*) ここで、*n* <= 21 マッピング・レベル 1.2 以上でサポートされる **注:** IBM Hexadecimal Floating Point (HFP) データ表記は、JSON に使用される IEEE-754-1985 表記とまったく同じではありません。一部の値が、ある表記から別の表記に正確に変換されない可能性があります。極端に大きいまたは小さい値は、float データ型に対して有効ではない場合があります。 HFP 表記との間の変換時に、一部の値で精度が失われる可能性があります。正確な変換が重要になる場合、BINARY FLOAT データ型を使用せずに、固定精度のデータ型を代わりに使用することを検討してください。	`"type":"number",` `"format":"float"`

PL/I のデータ記述	**JSON** スキーマ定義
`BINARY FLOAT( n )` ここで、`21 < n <= 53` 53 より大きい値はサポートされない。 マッピング・レベル 1.2 以上でサポートされる **注:** IBM Hexadecimal Floating Point (HFP) データ表記は、JSON に使用される IEEE-754-1985 表記とまったく同じではありません。一部の値が、ある表記から別の表記に正確に変換されない可能性があります。極端に大きいまたは小さい値は、float データ型に対して有効ではない場合があります。 HFP 表記との間の変換時に、一部の値で精度が失われる可能性があります。正確な変換が重要になる場合、BINARY FLOAT データ型を使用せずに、固定精度のデータ型を代わりに使用することを検討してください。	`"type":"number",` `"format":"double"`
`DECIMAL FLOAT( n )` ここで、`n <= 6` です。 マッピング・レベル 1.2 以上でサポートされる **注:** IBM Hexadecimal Floating Point (HFP) データ表記は、JSON に使用される IEEE-754-1985 表記とまったく同じではありません。一部の値が、ある表記から別の表記に正確に変換されない可能性があります。極端に大きいまたは小さい値は、float データ型に対して有効ではない場合があります。 HFP 表記との間の変換時に、一部の値で精度が失われる可能性があります。正確な変換が重要になる場合、DECIMAL FLOAT データ型を使用せずに、固定精度のデータ型を代わりに使用することを検討してください。	`"type":"number",` `"format":"float"`

PL/I のデータ記述	**JSON** スキーマ定義
DECIMAL FLOAT(*n*) ここで、6 < *n* <= 16 16 より大きい値はサポートされない。 マッピング・レベル 1.2 以上でサポートされる **注:** IBM Hexadecimal Floating Point (HFP) データ表記は、JSON に使用される IEEE-754-1985 表記とまったく同じではありません。一部の値が、ある表記から別の表記に正確に変換されない可能性があります。極端に大きいまたは小さい値は、float データ型に対して有効ではない場合があります。 HFP 表記との間の変換時に、一部の値で精度が失われる可能性があります。正確な変換が重要になる場合、DECIMAL FLOAT データ型を使用せずに、固定精度のデータ型を代わりに使用することを検討してください。	"type":"number", "format":"double"

name (*n*) *data description*	プリミティブの場合:
	
	"type":"array"
	"maxItems":
	n
	"minItems":
	n
	"items":{
	"type":"object",
	"properties":{
	name
	:{
	data description JSON
	}
	}
	"required":[
	name
	]
	}
	
	データ宣言の場合:
	
	"type":"array"
	"maxItems":
	n
	"minItems":
	n
	"items":{
	data description JSON
	}
	
	ここで、*data description JSON* は、PL/I データ記述の JSON スキーマ表記です。

JSON スキーマから **PL/I** へのマッピング**:**

DFHJS2LS ユーティリティー・プログラムは、JSON スキーマと PL/I データ構造との間のマッピングをサポートしています。Enterprise PL/I コンパイラーと古い PL/I コンパイラーの間には相違点があるため、PLI-ENTERPRISE と PLI-OTHER の 2 つの言語オプションがサポートされます。

PL/I にスキーマ・エレメント名をマップする場合の規則

CICS 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 PL/I 変数の固有で有効な名前を生成します。

1. A から Z、a から z、0 から 9、@、#、_、または $ 以外の文字は、「X」で置き換えられます。

   例えば、 monthly-total は monthlyXtotal となります。

   MAPPING-OVERRIDES パラメーターを使用して、他の文字の処理方法を変更できます。 例えば、値 **HYPHENS-AS-UNDERSCORES** を設定すると、JSON スキーマに含まれるハイフンは X ではなく下線に変換されます。例えば、monthly-total は monthly_total となります。
2. 同じスコープ内の重複した名前は、名前の 2 番目以降のインスタンスに 1 つまたは 2 つの数字を追加することによって固有にします。

   例えば、 year の 3 つのインスタンスは year 、 year1 、および year2 になります。

   もし上記の動作が望ましくないなら、ユーザーはユーティリティーの入力として MAPPING-OVERRIDES=NO-ARRAY-NAME-INDEXING を指定できます。これにより、名前の 2 番目以降のインスタンスへの 1 つまたは 2 つの数字の追加が無効になります。
3. JSON スキーマは、"type" 値が "array" で、キーワード "minItems" と "maxItems" が省略されるか、それぞれ別の値が指定されている場合には、変数の基数が可変になるように指定します。変数の基数が可変になるようスキーマで指定されている場合、フィールド名は接尾部 "_cont" および "_num" が付けられて作成されます。

   詳しくは、 324 ページの『DFHJS2LS 内のエレメントの可変配列』を参照してください。
4. JSON スキーマは、"required" キーワード配列 (周りを囲む JSON スキーマ "object" タイプに関連付けられている) に指定されていない変数がオプションとなるように指定します。オプション・フィールドの場合、追加のフィールドは、エレメント名に接尾部 _num が追加されて生成されます。実行時にこれは、JSON データに値が存在しなかったことを示す場合はゼロ、JSON データに値が存在した場合は非ゼロになります。
5. フィールド名は 31 文字までに制限されています。接頭部および接尾部を含めて生成された名前がこの長さを超えると、エレメント名が切り捨てられます。

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。

PL/I にスキーマ・タイプをマップする場合の規則

DFHJS2LS は、次の表に従ってスキーマ・タイプ値を PL/I データ・タイプにマップします。以下の点にも注意してください。

- **MAPPING-LEVEL** パラメーターを 1.2 以上に設定して、**CHAR-VARYING** パラメーターを NULL に設定すると、可変長文字データはヌル終了ストリングにマップされ、ヌル終止符として追加された 1 つの文字が割り振られます。
- **MAPPING-LEVEL** パラメーターを 1.2 以上に設定して、**CHAR-VARYING** パラメーターを指定しないと、可変長文字データはデフォルトでは、Enterprise PL/I の場合は VARYINGZ データ・タイプにマップされ、その他の PL/I の場合は VARYING データ・タイプにマップされます。
- 可変長バイナリー・データは、32 768 バイトより少ない場合は VARYING データ・タイプにマップされ、32 768 バイトを超える場合はコンテナーにマップされます。

JSON スキーマ・キーワード	**PL/I** のデータ記述

以下のすべて:
`"type":"null"`
`"type": []`
	サポートされていない

`"$schema":"http://json-schema.org/draft-04/schema#"`	このキーワードは無視されますが、ドラフト 04 JSON スキーマ仕様と互換性があることが想定されています。
`"title": "same text"` `"description": "more text"`	これらのキーワードは無視されます。
`"format": "<predefined values>"`	"format" キーワードは、生成された構造またはランタイム値のいずれかを変更するために使用されます。サポートされる "format" の使用方法については、以下を参照してください。

`"type":`
`"array",`
`"items": {<JSON Sub-schema>},`
`"additionalItems": false,`
`"maxItems":`
m
`,`
`"minItems":`
n
	多次元配列は、マッピング・レベル 4.3 以上でサポートされ、混合タイプ配列はサポートされません。
	
	"additionalItems" は、false になることが想定されています。他にサポートされている値はありません。
	
	"minItems" と "maxItems" の両方が存在し、それらが等しい場合、配列は固定基数として処理されます。それ以外の場合、配列は可変基数として処理されます。 324 ページの『DFHJS2LS 内のエレメントの可変配列』を参照してください。

JSON スキーマ・キーワード	**PL/I** のデータ記述
"type": "array", "uniqueItems": true	"uniqueItems" は、JSON 配列でサポートされていません。 <JSON Sub-schema> はサポートされている "type" を定義する必要がありますが、 "type" を "array" にすることはできません。 これは、生成される言語構造に対する制約事項です。
"type": "object", "additionalProperties": false, "properties": { ["<element name>": {<JSON Sub-schema>} [,]]* } "required": [["<element name>" [,]]*]	現在サポートされている JSON オブジェクトの唯一の形式は、名前付きエレメントの固定セットです。 これは、エレメント名を使用して構造 (または副構造) を生成します。 "additionalProperties" は、値が **ADDITIONAL-PROPERTIES-DEFAULT** パラメーターで設定されていない場合、false と想定されます。 "required" 配列にないか、"required" 配列が存在しない場合、"properties" オブジェクト内のエレメントはすべて "optional" と見なされます。 "optional" エレメントにはゼロから X までの可変オーディナリティーが提供されます (X は 1 または配列の項目の最大数、その項目は配列として定義)。 324 ページの『DFHJS2LS 内のエレメントの可変配列』を参照してください。

JSON スキーマ・キーワード	**PL/I** のデータ記述
`"type": "object",` `"additionalProperties": true`	オブジェクトは前の例のようにマップされ、追加プロパティーをサポートする追加フィールドがあります。 `"additionalProperties"` プロパティーは、**ADDITIONAL-PROPERTIES-DEFAULT** パラメーターで設定されていない場合、false と想定されます。 有効な場合、**ADDITIONAL-PROPERTIES-MAX** パラメーターで指定された値までスペースが割り当てられます。各スペースの文字数は、**ADDITIONAL-PROPERTIES-SIZE** パラメーターで設定されています。個々のプロパティーは、`PIC X(z)` フィールドにマップされます。*z* は、**ADDITIONAL-PROPERTIES-SIZE** パラメーターによって定義されます。 **ADDITIONAL-PROPERTIES-MAX** の値が 0 より大きい場合、プロパティーは、**maxItems** が設定された配列のプロパティーと同じようにマップされます。 **注:** CICS での z/OS Connect の使用など、CICS での JSON サポートを構成する方法はいくつかあります。古い CICS Java パイプライン・テクノロジーを使用する場合、**追加プロパティー**は、`com.ibm.cics.json.enableAxis2Handlers` JVM システム・プロパティーが true に設定されていないときにのみサポートされます。

`"type": "object",`
`"maxProperties": m,`
`"minProperties": n,`
`"patternProperties": {},`
`"dependencies":`
	これらのキーワードは、いずれも JSON オブジェクトではサポートされません。

`"type":"string"`
`"maxLength":`
m
`"pattern": "regular expression>",`
`"minLength":`
l
	`char[ z ]`
	
	ここで *z* の値は *m* に基づいていますが、 **CHAR-VARYING** パラメーターの設定に依存します。
	
	m は `"maxLength"` キーワードに基づき、固定長ストリングとして扱われます。
	
	`"pattern"` および `"minLength"` の制約事項は、コメントとしてのみ言語構造に渡されます。

JSON スキーマ・キーワード	**PL/I** のデータ記述
"type":"string" "maxLength": *m*	マッピング・レベル 4.0 以上で CCSID=1200 の場合: WIDECHAR(*z*) ここで *z* の値は *m* に基づいていますが、 **CHAR-VARYING** パラメーターの設定に依存します。 *m* は "maxLength" キーワードに基づき、固定長ストリングとして扱われます。

"*name*":{
"type":"string",
"format":"date-time"
}
	FIXED DECIMAL (15,0)
	
	DATETIME=PACKED15 の場合、すべてサポートされます。

"*name*":{
"type":"string",
"format":"uri"
}
	CHAR(*m*)
	
	ここで、m は "maxLength" キーワードに基づき、固定長ストリングとして扱われます。
	
	マッピング・レベル 4.0 以上で **CCSID=1200** の場合:
	
	WIDECHAR(*m*)
	
	ここで、*m* は "maxLength" キーワードに基づき、固定長ストリングとして扱われます。

"*name*":{
"type":"string",
"format":"base64Binary"
}
	CHAR(*m*)
	
	ここで、m は "maxLength" キーワードに基づきます。

"*name*":{
"type":"string",
"format":"hexBinary"
}
	CHAR(*m*)
	
	ここで、m は "maxLength" キーワードに基づきます。

JSON スキーマ・キーワード	**PL/I** のデータ記述

"*name*":{
"type":"string",
"format":"<predefined>"
}
	CHAR(*m*) ここで、*m* は "maxLength" キーワードに基づき、固定長ストリングとして扱われます。<predefined> は、*email*、*hostname*、*ipv4*、または *ipv6* のいずれかです。 関連する "pattern" がコメントに渡されます。
	
	マッピング・レベル 4.0 以上で **CCSID=1200** の場合:
	
	WIDECHAR(*m*)
	
	ここで、 *m* は "maxLength" キーワードに基づき固定長ストリングとして扱われます。 また、 <predefined> は *email*、*hostname*、 *ipv4*、 または *ipv6* のいずれかです。関連する "pattern" が使用され、コメントに渡されます。

"type":"boolean"	マッピング・レベル 1.1 以下: **Enterprise PL/I** SIGNED FIXED BINARY (7) その他の **PL/I** FIXED BINARY (7) マッピング・レベル 1.2 以上の場合 **Enterprise PL/I** BIT(7) BIT(1) その他の **PL/I** BIT(7) BIT(1) ここで、BIT(7) は位置合わせのために提供されており、BIT(1) にはブールでマップされた値が含まれます。
"type": "integer", "exclusiveMaximum": true, "exclusiveMinimum": true, "multipleOf": n	"exclusiveMaximum" および "exclusiveMinimum" の制約事項は、コメントとしてのみ言語構造に渡されます。 "multipleOf" は無視されます。
"type":"integer", minimum:-128, maximum:127	**Enterprise PL/I** SIGNED FIXED BINARY (7) その他の **PL/I** FIXED BINARY (7)

JSON スキーマ・キーワード	**PL/I** のデータ記述
"type":"integer", minimum:0, maximum:255	**Enterprise PL/I** UNSIGNED FIXED BINARY (8) その他の **PL/I** FIXED BINARY (8)
"type":"integer", minimum:-32768, maximum:32767	**Enterprise PL/I** SIGNED FIXED BINARY (15) その他の **PL/I** FIXED BINARY (15)
"type":"integer", minimum:0, maximum:65535	**Enterprise PL/I** UNSIGNED FIXED BINARY (16) その他の **PL/I** FIXED BINARY (16)
"type":"integer", minimum:-2147483648, maximum:2147483647	**Enterprise PL/I** SIGNED FIXED BINARY (31) その他の **PL/I** FIXED BINARY (31)
"type":"integer", minimum:0, maximum:4294967295	マッピング・レベル 1.1 以下: **Enterprise PL/I** UNSIGNED FIXED BINARY(32) マッピング・レベル 1.2 以上の場合 **Enterprise PL/I** CHAR(*y*) ここで *y* は 16 MB より小さい固定長です。 すべてのマッピング・レベル: その他の **PL/I** BIT(64)

JSON スキーマ・キーワード	**PL/I** のデータ記述

`"type":"integer",`
`minimum:-9223372036854775808,`
`maximum:9223372036854775807`
	マッピング・レベル 1.1 以下:
	Enterprise PL/I
	`SIGNED FIXED BINARY(63)`
	注: LIMITS コンパイラー・ディレクティブは、PL/I コンパイラーによるこのフィールドの解釈方法に影響する場合があります。CICS は、このフィールドが宣言されたサイズであることを想定しますが、コンパイラーはこのフィールドをより小さなスペースに最適化することがあるため、不一致が生じる可能性があります。このような問題を回避するには、LIMITS(FIXEDBIN(63)) コンパイル時オプションを使用します。
	マッピング・レベル 1.2 以上の場合
	Enterprise PL/I
	`CHAR(` *y* `)`
	ここで *y* は 16 MB より小さい固定長です。
	すべてのマッピング・レベル:
	その他の **PL/I**
	`BIT(64)`

JSON スキーマ・キーワード	**PL/I** のデータ記述

`"type":"integer",`
`minimum:0,`
`maximum:18446744073709551615`
	マッピング・レベル 1.1 以下:
	Enterprise PL/I
	`UNSIGNED FIXED BINARY(64)`
	注: LIMITS コンパイラー・ディレクティブは、PL/I コンパイラーによるこのフィールドの解釈方法に影響する場合があります。CICS は、このフィールドが宣言されたサイズであることを想定しますが、コンパイラーはこのフィールドをより小さなスペースに最適化することがあるため、不一致が生じる可能性があります。このような問題を回避するには、LIMITS(FIXEDBIN(63)) コンパイル時オプションを使用します。
	マッピング・レベル 1.2 以上の場合
	Enterprise PL/I
	`CHAR(` *y* `)`
	ここで *y* は 16 MB より小さい固定長です。
	その他の **PL/I**
	`BIT(64)`

`"type":"number"` `"description":"decimal"`	`FIXED DECIMAL(` *n* `,` *m* `)`

JSON スキーマ・キーワード	**PL/I** のデータ記述

`"type":"number"`
`"description":"float"`
	マッピング・レベル 1.0 および 1.1 の場合
	• `CHAR(32)`
	マッピング・レベル 1.2 以上の場合
	Enterprise PL/I
	`DECIMAL FLOAT(6) HEXADEC`
	その他の **PL/I**
	`DECIMAL FLOAT(6)`
	注: IBM Hexadecimal Floating Point (HFP) データ表記は、JSON に使用される IEEE-754-1985 表記とまったく同じではありません。一部の値が、ある表記から別の表記に正確に変換されない可能性があります。極端に大きいまたは小さい値は、float データ型に対して有効ではない場合があります。 HFP 表記との間の変換時に、一部の値で精度が失われる可能性があります。正確な変換が重要になる場合、DECIMAL FLOAT データ型を使用せずに、固定精度のデータ型を代わりに使用することを検討してください。

`"type":"number"`
`"description":"double"`
	マッピング・レベル 1.0 および 1.1 の場合
	• `CHAR(32)`
	マッピング・レベル 1.2 以上の場合
	Enterprise PL/I
	`DECIMAL FLOAT(16) HEXADEC`
	その他の **PL/I**
	`DECIMAL FLOAT(16)`
	注: IBM Hexadecimal Floating Point (HFP) データ表記は、JSON に使用される IEEE-754-1985 表記とまったく同じではありません。一部の値が、ある表記から別の表記に正確に変換されない可能性があります。極端に大きいまたは小さい値は、float データ型に対して有効ではない場合があります。 HFP 表記との間の変換時に、一部の値で精度が失われる可能性があります。正確な変換が重要になる場合、DECIMAL FLOAT データ型を使用せずに、固定精度のデータ型を代わりに使用することを検討してください。

JSON スキーマ・キーワード	**PL/I** のデータ記述

`oneOf: [ { "properties": { "A": {...} } } ,`
`  { "properties": { "B": {...} } } ]`
`anyOf: [ { "properties": { "A": {...} } } ,`
`  { "properties": { "B": {...} } } ]`
`allOf: [ { "properties": { "A": {...} } } ,`
`  { "properties": { "B": {...} } } ]`
	オプションの配列を使用した論理パスは、`{ "properties": { "A":..., "B":... } }` というタイプの単一オブジェクトが定義されている場合と同様にマージされます。JSON プロパティーはそれぞれオプションとして扱われます。同じプロパティー名の競合する定義がサブオプションに含まれている場合は、エラー・メッセージが発行されます。例えば、単一プロパティーがストリング (パス A 内) とオブジェクト (パス B 内) の両方として定義されている場合、そのような定義はサポートされず、エラー・メッセージが表示されます。
	
	JSON スキーマで複雑な論理構造を定義できますが、複雑な論理構造内で暗黙的に指定された細かい点は、言語構造へのマッピングで失われる可能性があります。変換プロセスは、スキーマからの組み合わせルールを強制しようとしません。指定された JSON プロパティーが存在するか存在しないかを示す言語構造フィールドとのみ対話します。
	
	サブオプションに同じプロパティー名で互換性のある定義が含まれている場合、DFHJS2LS は、制約の関連パターンをマージしようとしますが、プロセスではいくつかの細かい点が失われる場合があります。例えば、以下の定義を想定します。
	
	`"A": { "oneOf": [ { "type": "string", "maxLength": 5}, {"type": "string", "minLength": 7, "maxLength": 8} ] }`
	
	『A』 は、最大 5 文字の長さのストリング、または長さが 7 から 8 文字のストリングのいずれかとして定義されています。構成のマージにより、6 文字の長さが無効であることは認識されず、0 から 8 文字のストリングにマッピングされる可能性があります。
	
	論理構成を含むほとんどのシナリオでは単純な言語構造にマップされますが、複雑な論理構成では、マッピング処理中に妥協が図られることがあります。最良の結果を得るために、JSON スキーマで論理構成を使用せずに、同じ JSON プロパティーに代替宣言を定義してください。

注: CICS では、signed long の最大値 (2^63 - 1) より大きい整数値を変換できません。ただし、整数値が引用符で囲まれている場合は変換できます。

注: 数値型のスキーマに指定された最小値および最大値は、PL/I データ型にマップされる目的でのみ使用されます。実行時にデータがこれらの値に基づいて検証されることはありません。

変化するエレメントの配列

XML は、エレメント数が変化する配列を持つことができます。一般に、エレメント数が変化する WSDL 文書や XML スキーマは 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。 CICS はコンテナー・ベースのマッピングまたはインライン・マッピングを使用して XML におけるエレメント数の変化に対応します。

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。

- 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。

これは、アプリケーションの XML または 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。

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

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:length value="8"/>
</xsd:restriction>
</xsd:simpleType>
</xsd:element>
```

一般に、エレメント数が変化する WSDL 文書は 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 は一連のコンテナーとしてアプリケーション・プログラム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。これらの構造はアプリケーションへの入出力として使用されます。

- CICS が XML をアプリケーション・データに変換する場合、これらの構造にアプリケーション・データが追加され、アプリケーションがそのデータを読み取ります。
- CICS がアプリケーション・データを XML に変換する場合、アプリケーションによって構造内に追加されたアプリケーション・データが読み取られます。

これらのデータ構造のフォーマットは、一連の例を使用すると分かりやすくなります。 SOAP メッセージまたはアプリケーションの XML を使用できます。 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:length 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
```

マッピング・レベル 2 以下における変化するエレメント数

この例では、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:length value="8"/>
</xsd:restriction>
</xsd:simpleType>
</xsd:element>
```

主データ構造には、2 つのフィールドの宣言が格納されます。 CICS が XML をバイナリー・データに変換すると、最初のフィールドである component-num にはエレメントが XML 内に現れる回数が格納され、2 番目のフィールドである component-cont には、コンテナーの名前が格納されます。

```
05 component-num PIC S9(9) COMP-5
05 component-cont PIC X(16)
```

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)
```

エレメントが発生する回数を割り出すには、component-num (1 から 5 までの範囲の値が入ります) の値を検査する必要があります。 エレメント・コンテンツは

component-cont で指定されたコンテナー内にあります。このコンテナーはエレメントの配列を保持しており、各エレメントは DFHWS-component データ構造によってマップされています。

minOccurs="0" および maxOccurs="1" である場合、このエレメントはオプションです。 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、以下のように component-num 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 エレメントはなく、component-cont の内容は未定義です。
- この値が 1 の場合、component エレメントは component-cont 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。

注: SOAP メッセージが単一の繰り返しエレメントで構成される場合、DFHWS2LS は 2 つの言語構造を生成します。 主要な言語構造は、配列エレメントの数と、エレメントの配列を保持するコンテナーの名前を含んでいます。 2 番目の言語構造は、繰り返しエレメントの単一インスタンスをマップします。

マッピング・レベル 2.1 以上における変化するエレメント数

マッピング・レベル 2.1 以上では、CICS アシスタントで **INLINE-MAXOCCURS-LIMIT** パラメーターを使用できます。 **INLINE-MAXOCCURS-LIMIT** パラメーターは変化するエレメント数を処理する方法を指定します。 可変数のエレメント用のマッピング・オプションは、コンテナー・ベースのマッピング (390 ページの『マッピング・レベル 2 以下における変化するエレメント数』で説明されています) またはインライン・マッピングです。 このパラメーターの *value* は、0 - 32767 までの範囲の正整数です。

- **INLINE-MAXOCCURS-LIMIT** のデフォルト値は 1 であり、オプション・エレメントがインラインで確実にマップされます。
- **INLINE-MAXOCCURS-LIMIT** パラメーターに値 0 が設定されている場合、インライン・マッピングは使用されません。
- maxOccurs が **INLINE-MAXOCCURS-LIMIT** の値以下の場合、インライン・マッピングが使用されます。
- maxOccurs が **INLINE-MAXOCCURS-LIMIT** の値以上の場合、コンテナー・ベースのマッピングが使用されます。

変化するエレメント数をインラインでマップすると、配列 (上記の例では発生するオカレンスが固定) およびカウンターが生成されます。 component-num フィールドは存在するエレメントのインスタンス数を示し、それらのインスタンスは配列によって示されます。 390 ページの『マッピング・レベル 2 以下における変化するエレメント数』で示される例では、**INLINE-MAXOCCURS-LIMIT** が 5 以下の場合に生成されるデータ構造は次のようになります。

```
05 component-num PIC S9(9) COMP-5 SYNC.
05 component OCCURS 5 PIC X(8).
```

最初のフィールド (component-num) は前のセクションで示されたコンテナー・ベース・マッピングの例の出力と同じです。 2 番目のフィールドには長さが 5 の配列があり、生成される可能性のある最大エレメント数を収容できる大きさです。

インライン・マッピングは、エレメントの出現回数およびデータが収容されるコンテナーの名前を格納するコンテナー・ベースのマッピングとは異なり、現行のコンテナーにあるすべてのデータを格納します。 現行のコンテナーにデータを格納するとパフォーマンスが向上するので、インライン・マッピングの方が一般的には望ましいといえます。

ネストされた変数配列

複雑な WSDL 文書および XML スキーマには可変の繰り返しエレメントを含めることができ、そのエレメントにはさらに可変の繰り返しエレメントを含めることができます。 この場合、記述される構造は、例で説明した 2 つのレベルを超えます。

この例は、1 回から 5 回出現する可能性がある必須エレメント (<component1>) 内でネストされたオプションのエレメント (<component2>) を示します。

```
<xsd:element name="component1"
minOccurs="1" maxOccurs="5">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="component2"
minOccurs="0" maxOccurs="1">
<xsd:simpleType>
<xsd:restriction base="xsd:string">
<xsd:length value="8"/>
</xsd:restriction>
</xsd:simpleType>
</xsd:element>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
```

最上位のデータ構造は、前の例のデータ構造とまったく同じです。

```
05 component1-num PIC S9(9) COMP-5
05 component1-cont PIC X(16)
```

ただし、2 番目のデータ構造には、以下のエレメントが格納されます。

```
01 DFHWS-component1
02 component2-num PIC S9(9) COMP-5
02 component2-cont PIC X(16)
```

3 番目の構造には以下のエレメントが格納されます。

```
01 DFHWS-component2
02 component2 PIC X(8)
```

最外部エレメント <component1> の出現回数は、component1-num に格納されます。

component1-cont で指定されたコンテナーには、2 番目のデータ構造 (DFHWS-component1) のその数のインスタンスを持つ配列が含まれています。

component2-cont の各インスタンスは、異なるコンテナーを指定します。それぞれ、第 3 レベルの構造 (DFHWS-component2) によってマップされたデータ構造が含まれています。

この構造を説明するために、例と一致する次のような XML のフラグメントについて考えてみます。

```
<component1><component2>
string1
</component2></component1>
<component1><component2>
string2
</component2></component1>
<component1></component1>
```

<component1> は 3 回発生します。 最初の 2 つには、それぞれ <component2> のインスタンスが含まれていますが、3 番目のインスタンスには含まれていません。

最上位のデータ構造では、component1-num に値 3 が含まれています。 component1-cont で指定されたコンテナーには、DFHWS-component1 の次の 3 つのインスタンスがあります。

1. 第 1 のインスタンスでは、component2-num の値は 1 で、component2-cont で指定されたコンテナーは *string1* を保持します。
2. 第 2 のインスタンスでは、component2-num の値は 1 で、component2-cont で指定されたコンテナーは *string2* を保持します。
3. 第 3 のインスタンスでは、component2-num の値は 0 で、component2-cont の内容は未定義です。

このインスタンスでは、完全なデータ構造は、次の合計 4 つのコンテナーによって表現されます。

- コンテナー DFHWS-DATA 内のルート・データ構造
- component1-cont で指定されたコンテナー
- component2-cont の最初の 2 つのインスタンスで指定された 2 つのコンテナー

オプションの構造および xsd:choice

DFHWS2LS および DFHSC2LS はマッピング・レベル 2.1 以上で、<xsd:sequence>、<xsd:choice>、および <xsd:all> エレメントでの maxOccurs および minOccurs の使用をサポートします。この場合、minOccurs 属性および maxOccurs 属性は minOccurs="0" および maxOccurs="1" に設定されています。

アシスタントは、エレメント内の子エレメントがそれぞれオプションであるかのようにしてエレメントを処理するマッピングを生成します。 これらのエレメントを使用してアプリケーションを実装する場合、アプリケーションが無効なオプションの組み合わせを生成しないように確認してください。 それぞれのエレメントの生成された言語構造には count フィールドがありますが、これらのすべてが「0」に設定されるか、すべてが「1」に設定される必要があります。これ以外の値の組み合わせは、<xsd:choice> エレメントの場合を除き、無効です。

<xsd:choice> エレメントは、そのエレメントのオプションが 1 つのみ使用できることを示します。 これはすべてのマッピング・レベルでサポートされます。 アシスタントは <xsd:choice> のそれぞれのオプションを、minOccurs="0" および maxOccurs="1" である <xsd:sequence> エレメントのように扱います。 <xsd:choice> エレメントを使用してアプリケーションを実装する場合は、アプリケーションが無効なオプションの組み合わせを生成しないように確認してください。 それぞれのエレメントの生成された言語構造には count フィールドがありますが、そのうちの 1 つが「1」に、他のすべてが「0」に設定される必要があります。 こ

れ以外の組み合わせはすべて無効です。例外は、<xsd:choice> のエレメント自体がオプションの場合であり、すべてのフィールドが 0 に設定されていても有効です。

可変長の値と空白のサポート

CICS 支援機能の設定値を使用したり、XML スキーマに直接ファセットを追加したりすることにより、可変長の値と空白を処理する方法をカスタマイズできます。

通常、CICS XML 支援機能と CICS Web サービス支援機能は、データ・ストリングを固定長の文字配列にマップします。こうした配列では、スペースや NULL で埋め込む必要があります。 可変長の値を固定長のデータ配列にマッピングすると、効率が悪く、ストレージを浪費する恐れがあります。使用するデータ長が可変の場合、こうしたマッピングを処理する方法をカスタマイズすることが推奨されています。

ある言語構造から XML スキーマまたは WSDL 文書に変換する場合には、XML スキーマで `whiteSpace` ファセットと `maxLength` ファセットを指定し、支援機能で **CHAR-VARYING-LIMIT** パラメーターを設定することが推奨されています。

XML スキーマまたは WSDL 文書からある言語構造に変換する場合には、支援機能で **CHAR-VARYING** パラメーターを適切な値に設定することが推奨されています。

注: XML 文書では、NULL 文字 ('x00') は無効です。 CICS が構文解析するアプリケーション・データの NULL 文字はストリングの末尾を示すためのもので、値は切り捨てられます。 CICS は、アプリケーション・データを生成する際に、**CHAR-VARYING** パラメーターの値に従って生成します。 例えば、**CHAR-VARYING=NULL** オプションが指定されている場合、CICS によって生成される可変長のストリングはヌル文字で終わります。

XML から言語構造への可変長の値のマッピング

XML スキーマのファセットを使用して、または CICS 支援機能で特定のパラメーターを指定して、XML スキーマまたは WSDL 文書と言語構造間のマッピングを処理する方法をカスタマイズします。

ファセットを使用して XML データ・タイプを制限することができます。 XML 内の可変長データの処理方法をカスタマイズするには、長さファセット (`length`、`maxLength`、および `minLength`) および `whiteSpace` ファセットを使用します。

length
: データが固定長となるように指定します。

maxLength
: データ・タイプの最大長を指定します。 ストリング・ベースのデータ・タイプでこの値を設定しないと、最大長は無制限になります。

minLength
: データ・タイプの最小長を指定します。 ストリング・ベースのデータ・タイプでこの値を設定しないと、最小長は 0 になります。

whiteSpace
: データ値の周囲にある空白の処理方法を指定します。空白には、スペース、タブ、および改行が含まれます。 `whiteSpace` ファセットは、`preserve`、`replace`、または `collapse` に設定できます。

- 値 preserve は、データ値内の空白をすべて保持します。
- 値 replace は、タブまたは改行が適切な数のスペースに置換されることを意味します。
- 値 collapse は、先行空白、後続空白、および埋め込み空白は除去され、タブ、改行、および連続スペースはすべて単一のスペース文字に置き換えられることを意味します。

whiteSpace ファセットが設定されないと、空白は保持されます。

XML スキーマ・ファセットについて詳しくは、XML Schema Part 2: Datatypes Second Editionを参照してください。

CICS 支援機能でパラメーター DFHSC2LS および DFHWS2LS を使用すると、XML スキーマから言語構造への可変長データのマッピング方法を変更できます。これらのパラメーターは、マッピング・レベル 1.2 以降で使用可能です。

DEFAULT-CHAR-MAXLENGTH

XML スキーマ文書または WSDL 文書で長さが指定されていない場合に、マッピングする文字の文字データのデフォルトの配列長を指定します。 このパラメーターの値は、1 から 2 147 483 647 の範囲の正整数です。

ただし、XML スキーマまたは WSDL 文書で maxLength ファセットを使用して、DFHSC2LS または DFHWS2LS が使用する最大文字長を直接指定することをお勧めします。 XML スキーマまたは WSDL 文書で最大長を直接指定すると、すべてのストリング・ベースのデータ・タイプに 1 つのグローバル・デフォルトが適用されることに関連した問題を回避できます。

CHAR-VARYING-LIMIT

言語構造にマップされる可変長文字データの最大サイズを指定します。 文字データが、このパラメーターで指定されている値よりも大きい場合は、コンテナーにマップされ、生成された言語構造でそのコンテナー名が使用されます。値は 0 からデフォルトの 32 767 バイトの範囲で指定できます。

CHAR-VARYING

可変長文字データがどのようにマップされるのかを指定します。 このパラメーターを指定しない場合は、指定される言語に応じてデフォルトのマッピングが異なります。 以下のオプションを選択できます。

- **CHAR-VARYING=NO** は、可変長文字データが固定長ストリングとしてマップされるように指定します。
- **CHAR-VARYING=NULL** は、可変長文字データがヌル終了ストリングにマップされるように指定します。
- **CHAR-VARYING=YES** は、可変長文字データが PL/I で CHAR VARYING データ・タイプにマップされるように指定します。COBOL、C、および C++ 言語の場合、可変長文字データは、2 つの関連エレメント (データ長およびデータ) で構成される同等の表現にマップされます。

通常、**CHAR-VARYING=YES** に設定すると、最高のパフォーマンスが得られます。

言語構造から XML への可変長の値のマッピング

ご使用の言語構造と XML スキーマまたは WSDL 文書間におけるマッピングの処理方法をカスタマイズできます。 DFHLS2SC または DFHLS2WS の **CHAR-VARYING** パラメーターを COLLAPSE または NULL に設定して、文字配列が生成される方法を変更します。

CHAR-VARYING=NULL オプションを設定すると、CICS に XML を生成する際に各文字配列の末尾に NULL 文字を追加するように指示することになります。

CHAR-VARYING=COLLAPSE オプションを設定すると、CICS に XML を生成する際に文字配列の末尾から後続スペースを自動的に除去するように指示することになります。 このオプションが有効なのはマッピング・レベルが 2.1 以上の場合だけで、C および C++ 以外のすべての言語ではマッピング・レベル 2.1 以降のデフォルト値が **CHAR-VARYING=COLLAPSE** になります。 XML が解析されると、先行空白、後続空白、埋め込み空白はすべて削除されます。

アプリケーション・データでの UTF-16 サポート

CICS Web サービスでは、UTF-16 でエンコードされたアプリケーション・データから XML または JSON への変換と、XML または JSON から UTF-16 エンコード・アプリケーション・データへの変換がサポートされます。複数の言語でデータを保管および処理する必要がある場合には UTF-16 を使用してください。

CICS SOAP および JSON Web サービスでは、UTF-16 でエンコードされたアプリケーション・データから XML または JSON への変換と、XML または JSON から UTF-16 でエンコードされたアプリケーション・データへの変換がサポートされます。可変幅エンコード・スキームである Unicode を使用すると、システムは効率的にデータを扱うことができます。

UTF-16 は Unicode の可変幅エンコード方式であり、各文字が 2 バイトまたは 4 バイトで表されます。 CICS Web サービスはアプリケーション・データ用に CCSID 1200 をサポートします。これは、IBM 専用領域を含む UTF-16 BE (ビッグ・エンディアン) です。この動作は、サポートされるすべての言語で UTF-16 サポートと整合しています。

UTF-16 はマッピング・レベル 4.0 以上でサポートされます。アシスタントでマッピング設定を使用して、アプリケーション・データの変換方法をカスタマイズすることができます。 XML マッピング・レベルについて詳しくは、CICS アシスタントのマッピング・レベルを参照してください。JSON マッピング・レベルについて詳しくは、CICS JSON アシスタントのマッピング・レベルを参照してください。

注: UTF-16 は EBCDIC エンコード方式に比べてより多くの処理時間を必要とし、ストレージ効率が低くなります。さらに、複数の種類のエンコード方式を混合すると、実行時に追加の処理が発生します。

XML または JSON スキーマから言語構造への UTF-16 のマッピング

UTF-16 のサポートは、Web サービスの作成方法に応じて異なります。 XML または JSON スキーマから言語構造へのマッピング (トップダウン・マッピングともいう) には、次のような特性があります。 UTF-16 が有効になっている場合、すべてのテキスト・フィールドは UTF-16 フィールドにマップされ、COBOL での数値表

示データ型は EBCDIC としてマップされます。 UTF-16 を使用するには、DFHJS2LS、DFHSC2LS、または DFHWS2LS の CCSID パラメーターを 1200 に設定してください。

例えば、WSDL の中に以下の XML スキーマ断片が存在する場合、

```
<xsd:element name="myString" nillable="false">
<xsd:simpleType>
<xsd:restriction base="xsd:string">
<xsd:maxLength value="20"/>
</xsd:restriction>
</xsd:simpleType>
</xsd:element>
```

DFHWS2LS アシスタントは COBOL 言語構造で以下のフィールドを生成します。

```
myString PIC N(
20
) USAGE NATIONAL
```

Web サービス・アシスタントの CHAR-MULTIPLIER パラメーターを使用すると、アシスタントによって生成されるフィールドの長さを指定できます。

CHAR-MULTIPLIER

UTF-16 を使用する場合、 **CHAR-MULTIPLIER** パラメーターに有効な値は、2 または 4 のみで、 2 はデフォルト値です。

CHAR-MULTIPLIER = 2。 ここで、スキーマは maxlength *x* のストリングを表し、 PIC N(*x*) を生成します。 **CHAR-MULTIPLIER**=2 を設定した場合、UTF-16 ストリングでの代理ペアは使用不可にはなりませんが、フィールドに入る文字数が影響を受けます。

CHAR-MULTIPLIER = 4 は、PIC N(*2x*) を生成します。 **CHAR-MULTIPLIER**=4 である場合、1 つのエンコード・ユニットで表現できる文字がストリングに含まれるならば、実行時の値が埋め込まれます。

言語構造から XML または JSON スキーマへの UTF-16 のマッピング

言語構造から XML または JSON スキーマへのマッピング (ボトムアップ・マッピングともいう) は、トップダウン・マッピングとは異なる方法で管理されます。言語構造で UTF-16 ストリングが宣言されている場合、CICS はデータが UTF-16 でエンコードされていると解釈し、そうでない場合は EBCDIC エンコード方式のデータを想定します。 DFHLS2JS、DFHLS2SC、または DFHLS2WS の CCSID パラメーターは、アプリケーション・データ内の EBCDIC テキストのエンコードを示します (UTF-16 を示すようこれを設定してはなりません)。

UTF-16 文字として解釈されるデータ型は、COBOL では PIC N (*n*)、PL/I では WIDECHAR(*n*)、C および C++ では char16_t[*n*] です。

Web サービス・アシスタントの CHAR-USAGE パラメーターを使用して、データ型を指定できます。

CHAR-USAGE

COBOL では、各国語データ型 PIC N を UTF-16 または DBCS データ用に使用できます。この設定は NSYMBOL コンパイラー・オプションによって制御されます。データが適切に扱われるようにするには、アシスタントの

CHAR-USAGE パラメーターを NSYMBOL コンパイラー・オプションと同じ値に設定する必要があります。 UTF-16 を使用する場合には、通常、これは CHAR-USAGE=NATIONAL に設定されます。

同じコピーブック内で UTF-16 と DBCS のデータを含む各国語データ型を混合する必要がある場合は、個々のフィールドで USAGE NATIONAL または USAGE DISPLAY-1 修飾子を使用できます。

注: DFHLS2WS、DFHLS2SC、および DFHLS2JS は COBOL の GROUP USAGE NATIONAL 節をサポートしません。

Web サービス・アシスタントを使用した Web サービス・プロバイダーの作成

WSDL 1.1 または WSDL 2.0 に準拠する Web サービス記述から、または高水準言語データ構造から、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。 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を支援します。

このタスクについて

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。

サービス・プロバイダーと関連した Web サービス記述は、URI を使用して公開できます。 この URI は、WEBSERVICE と関連した URI と同じパスで、サフィックス ?wsdl が追加されたものになります。これにより、業務内のリクエスター、または業務外のリクエスターが、サービス・プロバイダーに関連した WSDL ファイルを見つけることができます。

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、WSDL 1.1 または WSDL 2.0 に準拠する 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。

始める前に

サービス・プロバイダー・アプリケーションを作成する前に、以下の条件が満たされていなければなりません。

- Web サービス記述が z/OS の UNIX ファイルに含まれている必要があり、適切なプロバイダー・モード・パイプラインを CICS 領域に作成する必要があります。
- 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、z/OS UNIX ライブラリーと PDS ライブラリーに対する読み取り権限と、**LOGFILE**、**WSBIND**、および **WSDL** 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。
- このユーザー ID で Java を実行するためには、この ID に十分なストレージを割り振る必要があります。 サポートされている任意のバージョンの Java を使用できます。デフォルトで、DFHWS2LS は **JAVADIR** パラメーターに指定されているバージョンの Java を使用します。

このタスクについて

Web サービス・アシスタントを使用して、WSDL の言語構造からサービス・プロバイダー・アプリケーションを作成することができます。 IBM WebSphere Service Registry and Repository (WSRR) サーバーに保管されている WSDL 文書を使用することもできます。

手順

1. Web サービス・バインディング・ファイルおよび 1 つ以上の言語データ構造を生成する場合は、DFHWS2LS バッチ・プログラムを使用します。 DFHWS2LS には、アプリケーションで必要なバインディング・ファイルと言語構造を作成する際に柔軟性を提供する、多数のオプション・パラメーターが含まれています。既存のアプリケーションを Web サービスで使用可能にするときは、以下のオプションを検討してください。
   a. サービス・プロバイダー・アプリケーション・プログラムにデータを渡すために **CICS** が使用すべきメカニズムは**?**
      - チャネルを使用してコンテナー内のデータを渡すか、COMMAREA を使用することができます。チャネルとコンテナーが推奨されます。 これは、**PGMINT** パラメーターを使用して指定します。

   b. 生成する言語は**?**
      - DFHWS2LS は、COBOL、C/C++、または PL/I 言語データ構造を生成できます。言語は、**LANG** パラメーターを使用して指定します。

   c. 使用するマッピング・レベルは**?**
      - マッピング・レベルが高いほど、実行時に文字とバイナリー・データの処理の制御とサポートを行いやすくなります。一部のオプション・パラメーターは、高いマッピング・レベルでしか使用できません。使用できる最高のマッピング・レベルを使用することをお勧めします。マッピング・レベルは **MAPPING-LEVEL** パラメーターで指定します。

   d. **Web** サービス・リクエスターで使用する **URI** は**?**

- **URI** パラメーターを使用して相対 URI を指定します。例えば、URI=/my/test/webservice です。この値は、URIMAP リソースの作成時に CICS によって使用されます。

e. **Web** サービス要求と応答を実行するためのトランザクションとユーザー **ID** は**?**

- 別名トランザクションを使用すると、サービス・リクエスターへの応答を構成するためのアプリケーションを実行することができます。別名トランザクションは、ユーザー ID を基にして接続されます。
- これを指定するには、**TRANSACTION** および **USERID** パラメーターを使用します。これらの値は、URIMAP リソースの作成時に使用されます。特定のトランザクションを使用したくない場合は、これらのパラメーターを使用しないでください。

f. **WSDL** 文書の保管場所は**?**

- ローカル・ファイル・システムからではなく、WSRR サーバーから WSDL 文書を取り出す場合は、DFHWS2LS でいくつかのパラメーターを指定する必要があります。
- 少なくとも、**WSRR-SERVER** パラメーターで WSRR サーバーの場所を指定し、 **WSRR-NAME** パラメーターでは WSRR から取り出す WSDL 文書の名前を指定する必要があります。
- WSRR を使用する場合に指定可能な他のパラメーターについては、553 ページの『DFHWS2LS: WSDL から高水準言語への変換』を参照してください。

g. **WSDL** 文書を **WSRR** サーバーから取り出す場合、セキュア接続を使用しますか**?**

- 適切なパラメーターを設定して Secure Sockets Layer (SSL) 暗号化を使用することにより、WSRR との間で安全に相互運用することができます。 例については、Web サービス・アシスタントと WSRR で SSL を使用する方法の例を参照してください。

- DFHWS2LS を実行依頼すると、CICS は Web サービス・バインディング・ファイルを生成して、ユーザーが **WSBIND** パラメーターを使用して指定した場所に置きます。言語構造は、ユーザーが **PDSLIB** パラメーターを使用して指定した区分データ・セットに置かれます。

2. 生成された 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プロバイダー・モード PIPELINE リソースの pickup ディレクトリーにコピーします。 バインディング・ファイルはバイナリー・モードでコピーする必要があります。
3. オプション: Web サービス記述、または 1 つ以上の Web サービス記述が含まれるアーカイブ・ファイルを、Web サービス・バインディング・ファイルと同じディレクトリーにコピーします。 アーカイブ・ファイルは .zip ファイルでなければならず、ファイル名は WSDL ファイル名と一致していなければなりません。 このコピーで、WSDL を見つけることができます。
4. 生成された言語構造とインターフェースを取るサービス・プロバイダー・アプリケーション・プログラムを作成して、必要なビジネス・ロジックを実装します。
5. WEBSERVICE リソースと 2 つの URIMAP リソースを作成します。

- WEBSERVICE リソースは、CICS で Web サービス・バインディング・ファイルをカプセル化し、実行時に使用されます。
- 最初の URIMAP リソースは、WEBSERVICE リソースを特定の URI に関連付けるための情報を CICS に提供します。
- 2 番目の URIMAP リソースは、WSDL アーカイブ・ファイルまたは WSDL 文書を特定の URI に関連付けるための情報を CICS に提供します。
  - この URI は、WEBSERVICE と関連した URI と同じパスで、サフィックス `?wsdl` が追加されたものになります。
  - この URIMAP リソースが作成されることにより、外部リクエスターがその URI を使用して、WSDL アーカイブ・ファイルまたは WSDL 文書を見つけることができます。
  - この URIMAP リソースが作成されるのは、Web サービス記述、または 1 つ以上の Web サービス記述が含まれるアーカイブ・ファイルが、Web サービス・バインディング・ファイルと同じディレクトリーにコピーされている場合だけです。
  - ピックアップ・ディレクトリーに WSDL アーカイブ・ファイルと WSDL 文書が含まれている場合、URI はアーカイブ・ファイル中の WSDL だけを返します。
  - この機能は、パイプライン走査操作を使用してインストールされた Web サービスのみで使用可能です。

以下の方法でリソースを作成できます。

a. **PIPELINE SCAN** コマンドを使用して、WEBSERVICE リソースと URIMAP リソースを動的に作成します。

b. リソースを自分で定義します。 CICS Explorer を使って CICS バンドル内で WEBSERVICE リソースを定義するときには、Web サービス・バインディング・ファイルと WSDL 文書または WSDL アーカイブ・ファイルをインポートして、それらをバンドルに含めるよう選択することができます。その後、Web サービスをサポートするための URIMAP 定義を生成して、これらをバンドル内にパッケージ化できます。 CICS Explorer を使用して CICS バンドル内のリソースを作成および編集する方法について詳しくは、CICS Explorer 製品資料内の『Working with bundles』を参照してください。

タスクの結果

DFHWS2LS の実行依頼時に問題が発生した場合や、リソースが正しくインストールされない場合は、配置エラーの診断を参照してください。

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。

始める前に

サービス・プロバイダー・アプリケーションを作成する前に、以下の前提条件がすべて満たされていることを確認してください。

- 高水準言語データ構造は、次の基準を満たす必要があります。
  - データ構造は、例えば COBOL コピーブック内など、ソース・プログラムとは別個に定義される必要があります。
  - PL/I または COBOL 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る必要があります。 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する必要があります。

    C および C++ では、区分データ・セット内の同じメンバーにデータ構造を置くことができます。
- ラッパー・プログラムを使用するかどうかに応じて、処理対象のデータ構造は次のように異なります。
  - 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。
- 言語構造が区分データ・セットで使用可能になっており、適切な PIPELINE リソースが CICS 領域に作成されていなければなりません。
  - 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、z/OS UNIX ライブラリーと PDS ライブラリーに対する読み取り権限と、**LOGFILE**、**WSBIND**、および **WSDL** 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。サポートされている任意のバージョンの Java を使用できます。デフォルトでは、DFHLS2WS は **JAVADIR** パラメーターで指定されている Java バージョンを使用します。

手順

データ構造を基にしたサービス・プロバイダー・アプリケーションを作成するには、以下のステップを実行します。

1. サービス・プロバイダー・アプリケーション・インターフェースがチャネルおよび多数のコンテナーを使用する場合には、XML でインターフェースを記述するチャネル記述文書を作成してください。 チャネル記述文書を z/OS UNIX 上の適切なディレクトリーに格納する必要があります。 CICS はこの文書を使用して、チャネル上のコンテナーから SOAP メッセージを構成および分解します。 あるいは、チャネル記述文書を作成せずに、チャネル上の 1 つのコンテナーを使用することもできます。
   - チャネル記述文書を作成する方法について、詳しくは、631 ページの『チャネル記述文書の作成』を参照してください。

2. 言語構造を基にして Web サービス・バインディング・ファイルおよび Web サービス記述を生成する場合は、DFHLS2WS バッチ・プログラムを使用します。 DFHLS2WS には、アプリケーションで必要なバインディング・ファイルと言語構造を作成する際に柔軟性を提供する、多数のオプション・パラメーターが含まれています。既存のアプリケーションを Web サービスとして使用可能にするために検討すべきオプションは、次のとおりです。

   a. サービス・プロバイダー・アプリケーション・プログラムにデータを渡すために **CICS** が使用すべきメカニズムは**?**

      - チャネルを使用してコンテナー内のデータを渡すか、COMMAREA を使用することができます。メカニズムは、**PGMINT** パラメーターを使用して指定します。アプリケーション・インターフェースがチャネルおよび多数のコンテナーを使用する場合には、**REQUEST-CHANNEL** パラメーターを指定し、オプションで **RESPONSE-CHANNEL** を指定します。これらのパラメーターは、マッピング・レベルが 3.0 以上の場合にのみ使用できます。

   b. 生成する **Web** サービス記述 **(WSDL** 文書**)** のレベルは**?**

      - CICS は、WSDL 1.1 文書または WSDL 2.0 文書と準拠する記述を生成します。両方のレベルの WSDL と準拠する要求をサービス・プロバイダー・アプリケーションでサポートするには、**WSDL_1.1** パラメーターと **WSDL_2.0** パラメーターに値を指定します。複数の WSDL パラメーターを使用する場合は、異なるファイル名になるようにします。 この仕様によって、2 つの Web サービス記述とバインディング・ファイルが生成されます。

   c. 使用する **SOAP** プロトコルのバージョンは**?**

      - **SOAPVER** パラメーターを使ってバージョンを指定できます。 ALL 値の使用をお勧めします。その場合、SOAP 1.2 メッセージ・ハンドラーで構成されたパイプラインに Web サービスをインストールしなければならなくなりますが、SOAP 1.1 または SOAP 1.2 のいずれかを Web サービス記述のバインディングとして柔軟に使用できます。このパラメーターを使用できるのは、**MINIMUM-RUNTIME-LEVEL** が 2.0 以上の場合だけです。

   d. 使用するマッピング・レベルは**?**

      - マッピング・レベルが高いほど、実行時に文字とバイナリー・データの処理の制御とサポートを行いやすくなります。一部のオプション・パラメーターは、高いマッピング・レベルでしか使用できません。**MAPPING-LEVEL** パラメーターで使用できる最も高いマッピング・レベルを指定することをお勧めします。

   e. **Web** サービス・リクエスターで使用する **URI** は**?**

      - **URI** パラメーターを使用して絶対 URI を指定します。例えば、**URI**=http://www.example.org:80/my/test/webservice です。このアドレスの相対部分 (/my/test/webservice) は、URIMAP リソースの作成時に使用されます。完全 URI は、Web サービス記述で <soap:address> エレメントとして使用されます。 この使用法は、HTTP URI と WebSphere MQ URI の両方に当てはまります。

   f. **WSDL** 文書を **IBM WebSphere Service Registry and Repository (WSRR)** に公開しますか**?**

- WSDL 文書を WSRR に公開する場合、DFHLS2WS で **WSRR-SERVER** パラメーターを指定する必要があります。 WSRR を使用する場合に指定できるパラメーターについて、詳しくは、 535 ページの『DFHLS2WS: 高水準言語から WSDL への変換』を参照してください。

g. **WSDL** 文書を **WSRR** サーバー上に公開する場合、セキュア接続を使用しますか**?**

- 適切なパラメーターを設定して Secure Sockets Layer (SSL) 暗号化を使用することにより、WSRR との間で安全に相互運用することができます。例については、Web サービス・アシスタントと WSRR で SSL を使用する方法の例を参照してください。

- DFHLS2WS を実行依頼すると、CICS は Web サービス・バインディング・ファイルを生成して、ユーザーが **WSBIND** パラメーターを使用して指定した場所に置きます。生成された Web サービス記述は、ユーザーが **WSDL**、**WSDL_1.1**、または **WSDL_2.0** パラメーターを使用して指定した場所に置かれます。
- DFHLS2WS で WSRR パラメーターを使用した場合には、指定された WSRR サーバーに WSDL 文書が公開されます。

3. 生成された Web サービス記述を確認して、必要なカスタマイズを行います。詳しくは、 633 ページの『生成した Web サービス記述文書のカスタマイズ』を参照してください。
4. 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プロバイダー・モード・パイプラインのピックアップ・ディレクトリーにコピーします。 Web サービス・バインディング・ファイルはバイナリー・モードでコピーする必要があります。
5. オプション: Web サービス記述、または 1 つ以上の Web サービス記述が含まれるアーカイブ・ファイルを、Web サービス・バインディング・ファイルと同じディレクトリーにコピーします。 アーカイブ・ファイルは .zip ファイルでなければならず、ファイル名は WSDL ファイル名と一致していなければなりません。 このコピーで、WSDL を見つけることができます。
6. **PIPELINE SCAN** コマンドを使用して、WEBSERVICE リソースおよび 2 つの URIMAP リソースを動的に作成します。 WEBSERVICE リソースは、CICS で Web サービス・バインディング・ファイルをカプセル化し、実行時に使用されます。
   - 最初の URIMAP リソースは、WEBSERVICE リソースを特定の URI に関連付けるための情報を CICS に提供します。
   - 2 番目の URIMAP リソースは、WSDL アーカイブ・ファイルまたは WSDL 文書を特定の URI に関連付けるための情報を CICS に提供します。
     - この URI は、WEBSERVICE と関連した URI と同じパスで、サフィックス `?wsdl` が追加されたものになります。
     - この URIMAP リソースが作成されることにより、外部リクエスターがその URI を使用して、WSDL アーカイブ・ファイルまたは WSDL 文書を見つけることができます。

- この URIMAP リソースが作成されるのは、Web サービス記述、または 1 つ以上の Web サービス記述が含まれるアーカイブ・ファイルが、Web サービス・バインディング・ファイルと同じディレクトリーにコピーされている場合だけです。
- ピックアップ・ディレクトリーに WSDL アーカイブ・ファイルと WSDL 文書が含まれている場合、URI はアーカイブ・ファイル中の WSDL だけを返します。
- この機能は、パイプライン走査操作を使用してインストールされた Web サービスのみで使用可能です。

別の方法として、リソースを自分で定義することもできますが、これは推奨されていません。

タスクの結果

CICS リソースを正常に作成したら、サービス・プロバイダー・アプリケーションの作成は完了です。

DFHLS2WS の実行依頼時に問題が発生した場合や、リソースが正しくインストールされない場合は、配置エラーの診断を参照してください。

次のタスク

サービスにアクセスする Web 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、この Web サービス記述を使用できるようにします。

チャネル記述文書の作成

サービス・プロバイダー・アプリケーションで多数のコンテナーを持つチャネル・インターフェースを使用する場合、チャネル記述文書を作成します。

このタスクについて

XML エディターを使用して、チャネル記述文書を作成します。チャネル記述のスキーマは、`channel.xsd` という名前で `/usr/lpp/cicsts/cicsts52/schemas/channel` ディレクトリーに保管されています (ここで、`/usr/lpp/cicsts/cicsts52` は CICS ファイルのデフォルト・インストール・ディレクトリーです)。

手順

1. 次のようにして、<channel> エレメントおよび CICS チャネル名前空間を持つ XML 文書を作成します。

```
<channel name="myChannel"
xmlns="http://www.ibm.com/xmlns/prod/CICS/channel">
</channel>
```

2. アプリケーション・プログラム・インターフェースがチャネルで使用するコンテナーごとに 1 つの <container> エレメントを追加します。 それぞれのコンテナーを記述する名前、タイプ、および使用属性を使用する必要があります。 以下の例は、異なる属性値を持つ 6 つのコンテナーを示しています。

```
<container name="cont1" type="char" use="required"/>
<container name="cont2" type="char" use="optional"/>
<container name="cont3" type="bit" use="required"/>
<container name="cont4" type="bit" use="optional"/>
```

```
<container name="cont5" type="bit" use="required">
<structure location="//HLQ.PDSNAME(MEMBER)"/>
</container>
<container name="cont6" type="bit" use="optional">
<structure location="//HLQ.PDSNAME(MEMBER2)"/>
</container>
```

この structure エレメントは、区分データ・セット・メンバーにある言語構造で内容が定義されていることを示します。

3. XML 文書を z/OS UNIX で保管します。

チャネル・スキーマ

チャネル記述文書は、以下のスキーマに従う必要があります。

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.ibm.com/xmlns/prod/CICS/channel"
xmlns:tns="http://www.ibm.com/xmlns/prod/CICS/channel"
elementFormDefault="qualified">
<element name="channel">
 1
<complexType>
<sequence>
<element name="container" maxOccurs="unbounded" "unbounded" minOccurs="0">
 2
<complexType>
<sequence>
<element name="structure" minOccurs="0">
 3
<complexType>
<attribute name="location" type="string" use="required"/>
<attribute name="structure" type="string" use="optional"/>
</complexType>
</element>
</sequence>
<attribute name="name" type="tns:name16Type" use="required"/>
<attribute name="type" type="tns:typeType" use="required"/>
<attribute name="use" type="tns:useType" use="required"/>
</complexType>
</element>
</sequence>
<attribute name="name" type="tns:name16Type" use="optional" />
</complexType>
</element>
<simpleType name="name16Type">
<restriction base="string">
<maxLength value="16"/>
</restriction>
</simpleType>
<simpleType name="typeType">
<restriction base="string">
<enumeration value="char"/>
<enumeration value="bit"/>
</restriction>
</simpleType>
<simpleType name="useType">
<restriction base="string">
<enumeration value="required"/>
<enumeration value="optional"/>
</restriction>
</simpleType>
</schema>
```

1. このエレメントは CICS チャネルを表します。
2. このエレメントはチャネル内の 1 つの CICS コンテナーを表します。

3. structure は「ビット」モードのコンテナーでのみ使用できます。 location 属性は、コンテナーの内容をマップするファイルの場所を示します。 structure 属性を C および C++ で使用して、構造体の名前を示すことができます。

次のタスク

DFHLS2WS を実行して、Web サービス・プロバイダー・アプリケーション用のマッピングと WSDL 文書を作成します。 DFHLS2WS は、コンテナーがチャネル記述文書で指定されている順序で、チャネルのマッピングを WSDL 文書に入れます。

生成した Web サービス記述文書のカスタマイズ

DFHLS2WS によって生成される Web サービス記述 (WSDL) 文書には、自動的に生成される内容が含まれていますが、これらは公開前にユーザーが変更した方がよい場合があります。 WSDL 文書をカスタマイズすると、Web サービス・バインディング・ファイルが再生成されることがあります。また、場合によっては、ラッパー・プログラムが作成されることもあります。

このタスクについて

以下の手順に従って、生成された Web サービス記述文書をカスタマイズします。

手順

1. HTTPS のサポートを公示するか、WebSphere MQ を使用して通信するには、DFHLS2WS で **URI** パラメーターを使用して、絶対 URI を設定します。 **URI** パラメーターを使用していない場合は、WSDL 文書の最後にある <wsdl:service> エレメントと <wsdl:binding> エレメントを変更する必要があります。 生成される WSDL には、これらの変更を行う際に役立つコメントが記載されています。これらのエレメントを変更するときに、Web サービス・バインディング・ファイルを再生成する必要はありません。
2. Web サービスのネットワークの場所を指定するには、DFHLS2WS で **URI** パラメーターを使用して、絶対 URI を設定します。 **URI** パラメーターを使用していない場合は、`wsdl:service` エレメント内の `soap:address` に詳細を追加します。
   a. HTTP ベースのプロトコルを使用する場合は、*my-server* を CICS 領域の TCP/IP ホスト名に置き換えて、*my-port* を TCPIPSERVICE リソースのポート番号に置き換えます。
   b. WebSphere MQ をトランスポート・プロトコルとして使用する場合は、*myQueue* を適切なキューの名前に置き換えます。

   これらの変更を行うために、Web サービス・バインディング・ファイルを変更する必要はありません。

   WSBind ファイルを再生成せずにポート名と名前空間を変更する場合、実行時レベル 2.1 以降では間違ったモニター情報になる可能性があります。
3. WSDL 文書内の自動的に生成された名前が目的に合っているかどうかを検討します。 以下の値を名前変更できます。
   - WSDL 文書の `targetNamespace`
   - WSDL 文書内の XML スキーマの `targetNamespace`

- <wsdl:portType> 名
- <wsdl:operation> 名
- <wsdl:binding> 名
- <wsdl:service> 名
- WSDL 文書内の XML スキーマにあるフィールドの名前

これらの値は、クライアント・プログラムをコーディングするプログラマチック・インターフェースの一部を形成します。 生成された名前に十分な意味がない場合、長期間にわたるアプリケーション・コードの保守が困難になる可能性があります。 XML スキーマの `targetNamespace` を変更するには、DFHLS2WS の **`REQUEST-NAMESPACE`** および **`RESPONSE-NAMESPACE`** パラメーターを使用し、WSDL 文書の `targetNamespace` を変更するには **`WSDL-NAMESPACE`** パラメーターを使用します。

これらの値のいずれかを変更する場合は、DFHWS2LS を使用して Web サービス・バインディング・ファイルを再生成する必要があります。生成される言語構造は、既存の言語構造と同じではありませんが、既存のアプリケーションとの互換性はあるため、アプリケーションを変更する必要はありません。 ただし、新規の言語構造を無視して、元の構造を持つ新規の Web サービス・バインディング・ファイルを使用することができます。

4. XML スキーマで公開されている COMMAREA フィールドが適切かどうかを検討します。 次のように、Web サービス・クライアント開発者にとって有益ではないフィールドを除去することができます。
   - 出力値のためにのみ使用されるフィールドを、入力データ構造をマップするスキーマから除去することができます。
   - 充てん文字フィールド。
   - 自動的に生成される注釈

   これらの変更を行う場合は、DFHWS2LS を使用して Web サービス・バインディング・ファイルを再生成する必要があります。 生成される新規の言語構造には、元の言語構造との互換性がないため、データを新規の表現から古い表現にマップするためのラッパー・プログラムを作成する必要があります。 このラッパー・プログラムは、ターゲット・アプリケーション・プログラムに対して **`EXEC CICS LINK`** コマンドを実行してから、戻されたデータをマップする必要があります。

   このレベルのカスタマイズは最も多くの労力を必要としますが、Web サービス・クライアント開発者にとって最も価値のあるプログラマチック・インターフェースにすることができます。
5. DFHWS2LS を介して、生成された WSDL 文書から新しい言語構造を作成する場合には、WSDL 文書内の注釈を保持するかどうかを決定してください。 DFHWS2LS が言語構造を生成するとき、注釈は通常のマッピング規則を指定変更します。マッピング規則を指定変更する際には、生成された言語構造と DFHLS2WS で使用されたバージョンとの互換性があることを確認してください。デフォルトのマッピング規則を使って言語構造を生成したい場合には、注釈を除去してください。

タスクの結果

カスタマイズされた WSDL 文書を IBM WebSphere Service Registry and Repository (WSRR) サーバーに公開するには、WSRR インターフェースを使って手動で公開する必要があります。 WSRR の詳細情報については、WebSphere Service Registry and Repositoryを参照してください。

例

WSDL 文書の例については、生成される WSDL 文書の例を参照してください。

SOAP 障害の送信

サービス・プロバイダーでは、CICS API を使用して、SOAP 障害を Web サービス・リクエスターに送信できます。 この障害は、サービス・プロバイダー・アプリケーション、またはパイプラインのヘッダー処理プログラムのいずれかによって発行されます。

始める前に

API を使用するには、サービス・プロバイダー・アプリケーションは、チャネルおよびコンテナーを使用する必要があります。アプリケーションが COMMAREA を使用する場合、チャネルおよびコンテナーを使用して SOAP 障害メッセージを作成するラッパー・プログラムを記述します。 CICS 提供の SOAP メッセージ・ハンドラーから直接 API が呼び出された場合、ヘッダー処理プログラムでのみ API を使用できます。

このタスクについて

ご使用のアプリケーション・ロジックでは要求を満たすことができない場合 (要求メッセージに根本的な問題が存在する場合など)、Web サービス・リクエスターに SOAP 障害を発行します。 CICS は、SOAP 障害の発行をエラーとして見なさないので、エラー処理ではなく、標準的なメッセージ応答のパイプライン処理が実行されることに注意してください。 いずれかのトランザクションをロールバックするには、アプリケーション・プログラムを使用する必要があります。

手順

1. プログラムで、 **EXEC CICS SOAPFAULT CREATE** コマンドを使用して、SOAP 障害を送信します。
2. コマンドに CLIENT または SERVER オプションを追加します。 このオプションは、クライアント側またはサーバー側のいずれかで、問題が発生したことを示します。
   - CLIENT は、受信された要求メッセージに問題があることを示します。
   - SERVER は、要求メッセージが CICS で処理されるときに問題が発生したことを示します。 これは、アプリケーション・プログラムの問題である可能性があります (例えば、要求を満たすことができない可能性がある)。あるいは、パイプライン処理中に発生する内在的な問題である可能性もあります。
3. サービス・プロバイダーによって障害が発行された理由を要約する FAULTSTRING オプションと、その長さを表す FAULTSTRLEN オプションを追加します。 このオプションの内容は、XML でなくてはなりません。 アプリケーシ

ョンが提供するすべてのデータは、XML 文書に直接含めるのに適した形式でなければなりません。 アプリケーションは、いくつかの文字を XML エンティティーとして指定する必要があるかもしれません。 例えば、XML タグの始まり以外の場所で < 文字が使われる場合、アプリケーションはそれを < に変更する必要があります。 以下の例は、間違った FAULTSTRING オプションを示しています。

```
dcl msg_faultString char(*) constant('Error: Value A
< Value B');
```

この FAULTSTRING オプションを指定する正しい方法は、次のとおりです。

```
dcl msg_faultString char(*) constant('Error: Value A
&lt; Value B');
```

ヒント: XML エンティティーの使用を避けるために、XML CDATA 構成体でデータをラッパーすることができます。 XML プロセッサーは、この構成体の中の文字データを構文解析しません。 この方式を使って、次のような FAULTSTRING オプションを指定できます。

```
dcl msg_faultString char(*)
constant('<![CDATA[Error: Value A < Value B]]>');
```

4. サービス・プロバイダーによって障害が発行された理由を詳しく記述する DETAIL オプションと、その長さを表す DETAILLENGTH オプションをコーディングします。 このオプションの内容は、XML でなくてはなりません。 FAULSTRING オプションと同じ指針が DETAIL オプションにも該当します。
5. オプション: ヘッダー処理プログラムから API を起動した場合は、パイプライン構成ファイルでそのプログラムを定義します。 ヘッダー処理プログラムは、<cics_soap_1.1_handler>、<cics_soap_1.2_handler>、<cics_soap_1.1_handler_java>、または <cics_soap_1.2_handler_java> エレメントのいずれかで定義されます。

タスクの結果

プログラムがこのコマンドを発行した場合、CICS は、該当する SOAP レベルで SOAP 障害の応答メッセージを作成します。サービス・プロバイダー・アプリケーションがコマンドを発行した場合は、SOAP 応答を作成する必要はなく、コマンドを DFHRESPONSE コンテナーに入れます。パイプラインは、メッセージ・ハンドラーによって SOAP 障害を処理し、その応答を Web サービス・プロバイダーに送信します。

例

SOAPFAULT CREATE コマンドには多くのオプションがあり、Web サービス・リクエスターに適切に応答できる柔軟性を提供します。 次に示す簡単な例は、SOAP 障害を作成する、SOAP 1.1 および SOAP 1.2 の両方で使用できる完全なコマンドです。

```
EXEC CICS SOAPFAULT CREATE CLIENT
DETAIL(
msg_detail
)
DETAILLENGTH(length(
msg_detail
))
FAULTSTRING(
```

```
                    msg_faultString
                    )
                    FAULTSTRLEN(length(
                    msg_faultString
                    ));
```

msg_detail および *msg_faultString* には、以下の値をコーディングできます。

```
dcl msg_detail char(*)
    constant('<ati:ExampleFault xmlns="http://www.example.org/faults"
    xmlns:ati="http://www.example.org/faults">Detailed error message
    goes here.</ati:ExampleFault>');
    dcl msg_faultString char(*) constant('Something went wrong');
```

Web サービス・アシスタントを使用した Web サービス・リクエスターの作成

WSDL 1.1 または WSDL 2.0 に準拠する Web サービス記述から、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。 CICS Web サービス・アシスタントは、サービス・リクエスター設定での CICS アプリケーションの配置を支援します。

始める前に

Web サービス記述が z/OS UNIX のファイルに存在するか、IBM WebSphere Services Registry and Repository (WSRR) サーバー上に公開される必要があります。さらに、リクエスター・モードのパイプラインが CICS 領域にインストールされている必要があります。

このユーザー ID で Java を実行するためには、この ID に十分なストレージを割り振る必要があります。 サポートされている任意のバージョンの Java を使用できます。デフォルトで、DFHWS2LS は **JAVADIR** パラメーターに指定されているバージョンの Java を使用します。

このタスクについて

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必要があります。

手順

1. Web サービス・バインディング・ファイルおよび 1 つ以上の言語構造を生成する場合は、DFHWS2LS バッチ・プログラムを使用します。 Web サービス記述からサービス・リクエスター・アプリケーションを作成する場合、次のようなオプションを考慮してください。
   - 使用するマッピング・レベルは? マッピング・レベルが高いほど、実行時に文字とバイナリー・データの処理の制御とサポートを行いやすくなります。一部のオプション・パラメーターは、高いマッピング・レベルでしか使用できません。使用できる最高のマッピング・レベルを使用することをお勧めします。

- 実行時にデータを変換する際に使用するコード・ページは? CICS 領域のコード・ページと異なる特定のコード・ページをアプリケーションに使用する場合は、**CCSID** パラメーターを使用します。
- Web サービス記述で宣言された操作のサブセットをサポートしますか? Web サービス記述が非常に大きい場合、特定の数の操作だけをサービス・リクエスター・アプリケーションでサポートするには、**OPERATION** パラメーターを使って必要な操作をリストします。 それぞれの操作はスペースで区切る必要があり、大/小文字が区別されます。
- WSDL 文書の保管場所は? DFHWS2LS への入力として使用する WSDL 文書が WSRR サーバーに格納されている場合、いくつかのパラメーターを指定して DFHWS2LS を実行することにより、これを取り出すことができます。 **WSRR-SERVER** パラメーターを使って WSRR サーバーの場所を指定し、**WSRR-NAME** パラメーターを使用して、取り出す WSDL 文書の名前を指定します。 WSRR との対話で使用できる DFHWS2LS の他のパラメーターについては、 553 ページの『DFHWS2LS: WSDL から高水準言語への変換』を参照してください。
- WSDL 文書を WSRR サーバーから取り出す場合、セキュア接続を使用しますか? Web サービス・アシスタントで Secure Sockets Layer (SSL) 暗号化を使用して、安全に WSRR と相互運用することができます。 例については、Web サービス・アシスタントと WSRR で SSL を使用する方法の例を参照してください。

DFHWS2LS の使用時には、**PROGRAM**、**URI**、**TRANSACTION**、および **USERID** などのパラメーターは指定しないでください。 これらのパラメーターは、サービス・プロバイダー・アプリケーションのみに適用されます。指定すると、プロバイダー・モードの Web サービス・バインディング・ファイルが生成されます。Web サービス・バインディング・ファイルに加えて、このプログラムは言語データ構造を生成します。

2. ログ・ファイルを調べて、DFHWS2LS がバインディング・ファイルおよび言語構造を生成したときに問題が発生したかどうかを確認します。 Web サービス記述には、CICS でサポートされないエレメントやオプションが含まれる可能性があります。 警告またはエラー・メッセージが出される場合、記されているアドバイスを読んで適切な処置を行ってください。 バッチ・プログラムを再実行しなければならないことがあります。
3. 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リクエスター・モード・パイプラインのピックアップ・ディレクトリーにコピーします。
4. PIPELINE リソースがサービス・リクエスター・アプリケーションに対して構成されていることを確認します。 **MODE** パラメーターの値は、パイプラインがサポートするのがリクエスター Web サービス・アプリケーションかプロバイダー Web サービス・アプリケーションかを示します。
5. 正しい SOAP プロトコルが Web サービスのパイプラインでサポートされていることを確認します。 **SOAPLEVEL** パラメーターは、サポートされるバージョンを示します。 サービス・リクエスター・モードでは、Web サービスのバインディングは、パイプラインでサポートされる SOAP のバージョンと一致している

必要があります。 SOAP 1.2 をサポートするサービス・リクエスター・パイプラインに SOAP 1.1 バインディングを使用する Web サービスをインストールすることはできません。

6. パイプラインの構成済みタイムアウトがサービス・リクエスター・アプリケーションに適していることを確認します。 タイムアウトは、PIPELINE リソースで RESPWAIT 属性の値として表示されます。パイプラインでタイムアウトが指定されていない場合は、トランスポートのデフォルトが使用されます。
   - HTTP のデフォルト・タイムアウトは 10 秒です。
   - WebSphere MQ のデフォルトのタイムアウトは 60 秒です。

   CICS 領域内のトランザクションごとにディスパッチャー・タイムアウトがあります。 この値がいずれかのプロトコルのデフォルトより小さい場合、ディスパッチャーによってタイムアウトになります。
7. オプション: Web サービス記述を、Web サービス・バインディング・ファイルと同じピックアップ・ディレクトリーにコピーします。これにより、実行時の Web サービスの検証をオンにすることができます。
8. WEBSERVICE リソースを作成します。このリソースは、CICS で Web サービス・バインディング・ファイルをカプセル化し、実行時に使用されます。

   以下の方法でこれを行うことができます。

   a. **PIPELINE SCAN** コマンドを使用して、WEBSERVICE リソースを動的に作成します。
   b. 自分でリソースを定義します。 CICS Explorer を使って CICS バンドル内で WEBSERVICE リソースを定義するときには、Web サービス・バインディング・ファイルと WSDL 文書または WSDL アーカイブ・ファイルをインポートして、それらをバンドルに含めるよう選択することができます。その後、Web サービスをサポートするための URIMAP 定義を生成して、それらをバンドル内にパッケージ化できます。 CICS Explorer を使用して CICS バンドル内のリソースを作成および編集する方法について詳しくは、CICS Explorer 製品資料内の『Working with bundles』を参照してください。
9. 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。 ラッパー・プログラムを作成する場合は、ステップ 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 SERVICE** コマンドを使用します。 以下のオプションがコマンドに含まれます。
   - Web サービスの URI
   - Web サービスの呼び出し対象となる操作

   Web サービスを呼び出す際に、その Web サービスの URI に関する情報を格納する URIMAP リソースを指定できます。 URIMAP リソースを使用する代わりに、INVOKE SERVICE コマンドで直接この情報を指定できます。ただし、URIMAP リソースを使用した場合は、サービス・プロバイダーの URI が変更されてもアプリケーションを再コンパイルする必要がありません。
   URIMAP リソースによって、接続プールを実装することもできます。接続プールでは、CICS は使用後のクライアント接続をオープンしたままにして、同じアプリケーションが以降の要求で再利用したり、同じサービスを呼び出す別のアプ

リケーションが再利用したりできます。 PIPELINE SCAN コマンドは、サービス・リクエスターの URIMAP リソースを作成しません。そのため、CICS の URIMAP リソースを HTTP クライアントとして作成の指示に従って、URIMAP リソースを自分で定義する必要があります。

タスクの結果

CICS リソースを正常に作成したら、サービス・リクエスター・アプリケーションの作成は完了です。

ツールを使用した Web サービスの作成

Web サービス・アシスタント JCL を使用する代わりに、IBM Developer for Z を使用するか、独自の Java プログラムを作成して、CICS で必要なファイルを作成することができます。

手順

1. 以下の 2 つの選択肢があります。
   - IBM Developer for Z ツールを使用して、Web サービス・バインディング・ファイル、および Web サービス記述または言語構造を作成します。 このツールについて詳しくは、IBM Developer for z Systems を参照してください。
   - 提供されている API を使用して独自の Java プログラムを作成して、Web サービス・アシスタントを起動します。 この API については『Reference』の『Application development』の Web サービス・アシスタント Javadoc 情報 Javadoc で説明されています。この資料には、クラスについて説明したコメントが記載されており、Web サービス・アシスタントの起動方法の例を示したサンプル・コードも提供されています。 また、Javadoc には、必要な JAR ファイルと、z/OS UNIX でのそれらの場所を示す詳細なリストも含まれています。

     Java プログラムは z/OS、Windows、または Linux プラットフォーム上で実行できます。 Windows または Linux でこのプログラムを実行する場合は、FTP または同等のプロセスを使用して、生成済みの Web サービス・バインディング・ファイルをバイナリー・モードで適切なピックアップ・ディレクトリーに転送します。
2. オプション: Web サービス記述を言語構造から生成する場合は、ファイルを検討し、必要なカスタマイズを実行します。 詳しくは、 633 ページの『生成した Web サービス記述文書のカスタマイズ』を参照してください。
3. 生成された Web サービス・バインディング・ファイルを、適切なパイプライン・ピックアップ・ディレクトリーに配置します。
4. オプション: Web サービス記述をパイプラインのピックアップ・ディレクトリーにコピーして、これによって、Web サービスの検証を実行して Web サービスが予想どおりに動作していることをチェックすることもできます。
5. Web サービス記述から始めた場合は、生成された言語構造とインターフェースをとるサービス・プロバイダーまたはリクエスターのアプリケーション・プログラムを作成します。

6. **PIPELINE SCAN** コマンドを実行して、必要な CICS リソースを自動的に作成します。

XML を認識する独自の Web サービス・アプリケーションの作成

CICS 提供のデータ・マッピングを使用しない場合、その代わりに、XML を認識するデータ・アプリケーションを 2 つの方法で独自に作成できます。 DFHWS2LS で **XML-ONLY** パラメーターを使用するか、あるいはツールをまったく使用せずに独自にアプリケーションを作成することができます。 **XML-ONLY** パラメーターを使用する方法は、XML を処理対象データとしてアプリケーションに渡すよう CICS パイプライン・プロセスを構成する最も単純な方法です。

このタスクについて

XML を認識する独自のアプリケーションを作成するには、XML 文書を構文解析するコードと生成するコードを作成する必要があります。 XML を認識する独自のアプリケーションを作成する 1 つの方法は、COBOL で `XML PARSE` および `XML GENERATE` ステートメントを使用することです。 XML を認識する独自のアプリケーションを作成する別の方法として、他の IBM ツールを使用できます。例えば、IBM Developer for Z ツールを使用すると、アプリケーションから起動可能な COBOL XML コンバーター・プログラムを生成できます。

XML 認識サービス・プロバイダー・アプリケーション

XML を認識する独自のサービス・プロバイダー・アプリケーションは、渡されるコンテナーと連携して、XML とプログラム言語の間のデータ変換を処理する必要があります。

このタスクについて

以下の手順に従うと、XML 認識アプリケーションを独自に作成できます。その際、いずれかの CICS ツールを使用するかどうかを決定できます。

手順

1. DFHWS2LS を使って独自の XML 認識アプリケーション用の Web サービス・バインディング・ファイルを生成するかどうかを決定します。 Web サービス・バインディング・ファイルを生成する利点は、検証などの CICS サービスを使用してグローバル・ユーザー出口を使用する Web サービスや CICS モニターをテストできることです。
   - Web サービス・バインディング・ファイルを生成するには、**XML-ONLY** パラメーター、および 2.1 以上の **MINIMUM-RUNTIME-LEVEL** を指定して DFHWS2LS を実行します。 Web サービス・バインディング・ファイルにより、アプリケーション・プログラムは DFHWS-BODY コンテナーの内容を直接処理できるようになります。これ以外のすべての点では、生成されるバインディング・ファイルの配置特性と実行時の動作は、**XML-ONLY** パラメーターなしで生成されるファイルと同じです (SOAP メッセージ処理中の XML 解析も含めて)。この解析を回避するには、パイプライン構成ファイル内に SOAP メッセージ・ハンドラー を指定しないようにする必要があります。

- Web サービス・バインディング・ファイルを使用しない場合、Web サービス要求が XML 認識アプリケーションに到達するようにサービス・プロバイダー・パイプラインを構成してください。 独自の XML 認識アプリケーション・プログラムを使用するようにパイプライン構成ファイル内で端末ハンドラーを構成できます。または、パイプラインで受信される URI に応じて独自のアプリケーションに動的に切り替わるメッセージ・ハンドラーを作成することもできます。

2. 以下のコンテナーで保持される Web サービス要求を処理するように、アプリケーションを作成します。

   DFHWS-BODY
   CICS 提供の SOAP メッセージ・ハンドラーがパイプラインに含まれる場合のインバウンド SOAP 要求に関する、SOAP 本体の内容。

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

   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。

   API コマンドをコーディングしてコンテナーで作業する場合に、CHANNEL オプションの指定はありません。コンテナーはすべて現在のチャネル (プログラムに渡されたチャネル) に関連付けられているためです。チャネルの名前を知りたい場合は、**EXEC CICS ASSIGN CHANNEL** コマンドを使用します。

3. オプション: アプリケーションでは、パイプライン内のメッセージ・ハンドラーで使用可能な追加のコンテナーを使用することもできます。また、メッセージ・ハンドラーの処理時に作成される他の任意のコンテナーを使用することもできます。 コンテナーの詳細なリストについては、パイプラインで使用されるコンテナーを参照してください。

4. アプリケーションが要求を処理したら、以下のコンテナーを使用して、Web サービス応答を構成します。

   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エンベロープを含む完全な SOAP メッセージを作成する場合、CICS が提供する SOAP メッセージ・ハンドラーを使用する代わりに、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 は無視されます。

   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。

要求と応答が同一であっても、プログラムでこのコンテナーを作成する必要があります。作成しないと、CICS が内部サーバー・エラーを発行します。

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もできます。

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。

5. URIMAP 応答を作成します。 **XML-ONLY** パラメーターを使用し、しかも DFHWS2LS の **URI** パラメーターの値を指定した場合には、PIPELINE SCAN 処理中に URIMAP が自動的に作成されます。

XML 認識サービス・リクエスター・アプリケーションの作成

XML を認識する独自の Web サービス・リクエスター・アプリケーションは、XML とプログラミング言語の間のデータ変換を処理し、パイプライン内の制御コンテナーにデータを設定します。

始める前に

DFHWS2LS で **XML-ONLY** パラメーターを使用することにより、独自の XML 認識サービス・リクエスター・アプリケーションを作成できます。または、ツールをまったく使用せずに作成することもできます。 独自の XML 認識サービス・リクエスター・アプリケーションを作成する最も単純な方法は、DFHWS2LS で **XML-ONLY** パラメーターを使用する方法です。 **XML-ONLY** パラメーターは実行時レベル 2.1 以降で使用できます。

このタスクについて

XML-ONLY パラメーターを使用した結果として生成される WSBind ファイルは、アプリケーションが DFHWS-BODY コンテナーの内容を直接処理することを CICS に対して示します。 リクエスター・モードの WEBSERVICE リソースを作成するために、生成された WSBind ファイルはリクエスター・モードの PIPELINE にインストールされる必要があります。アプリケーションは Web サービス要求の本体用の XML を生成して、それを DFHWS-BODY コンテナーに格納する必要があります。その後、**EXEC CICS INVOKE SERVICE** コマンドを呼び出す必要があります。アウトバウンド・メッセージが Web サービス・プロバイダーに送られます。呼び出しが完了した後、応答メッセージの本文もまた DFHWS-BODY コンテナーに入ります。

応答メッセージの XML は、SOAP メッセージ処理中に解析されます。この解析を回避するには、パイプライン構成ファイル内に SOAP メッセージ・ハンドラー を指定しないようにする必要があります。

XML を認識するリクエスター・アプリケーションは、リモート・プロバイダー・モード・アプリケーションから戻される SOAP 障害メッセージを受け取ることがあります。この場合、リクエスター・アプリケーションは SOAP 障害を解釈して、それを通常の応答メッセージと区別する必要があります。 **INVOKE SERVICE** コマンドを

XML-ONLY WEBSERVICE と共に使用する場合、CICS は、SOAP 障害の受信を示すために通常使われる応答コードを設定しません。

XML-ONLY オプションを使用せずに独自の XML 認識サービス・リクエスター・アプリケーションを作成する場合は、以下の手順を完了してください。

手順

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。 制御コンテナーはすべて CHAR モードで取り込まれなければなりません。 各コンテナーに次の情報を指定します。

   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エンベロープを含む完全な SOAP メッセージを作成する場合、このコンテナーを使用します。 アウトバウンド・メッセージで SOAP ヘッダーが重複して送られるのを防ぐために、CICS 提供の SOAP メッセージ・ハンドラーがパイプラインに含まれていてはなりません。

   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。

   ヒント**:** コンテナー DFHWS-NOABEND をチャネルに追加すると、DFHPIRT の呼び出し時に DFHPIRT 内部から異常終了が発行されることはありません。DFHERROR コンテナーを介してエラーを処理できるため、C/C++ プログラムを実行する場合にはこれが役立ちます。

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* はコンテナーを保持するチャネルです。 アウトバウンド・メッセージは、パイプライン内のメッセージ・ハンドラーおよびヘッダー処理プログラムによって処理され、Web サービス・プロバイダーに送られます。

3. 同じチャネルからの Web サービス応答を格納するコンテナーを取り出します。Web サービス・プロバイダーからの応答は、成功した応答か SOAP 障害の可能性があります。 Web サービス・リクエスター・アプリケーションは、サービス・プロバイダーからのいずれのタイプの応答も処理できる必要があります。次のコンテナーに完全な応答が格納されます。

   DFHRESPONSE
   SOAP 応答のエンベロープを含む Web サービス・プロバイダーから受信した完全な応答。

   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。

   注**:** エラーのケースによっては、DFHWS-BODY が更新されない場合があります。SOAP 障害について DFHRESPONSE を確認する必要があります。

Web サービスでの Java の使用

Java を使用して、Web サービス・アプリケーションを作成できます。これらのアプリケーションの作成には、他のプログラミング言語で使用する技法とは異なる技法が使用されます。

ほとんどの非 Java プログラミング言語では、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有効にします。 Web サービス・アシスタントを使用するということは、 CICS が Web サービスのデータをアプリケーションに適した形式に変換し、それをコンテナーまたは COMMAREA に配置することを意味します。Java アプリケーションに Web サービス・アシスタントを使用することもできますが、Java アプリケーション用の Java Web サービスを作成する方法としては、以下のタスクがより適しています。

Axis2 JVM サーバーにおける Java プロバイダー・モードの Web サービスの配置

CICS で、Axis2 アプリケーションをプロバイダー・モード Web サービスとして配置できます。これらのアプリケーションは通常 JAX-WS を使用して生成され、Java 有効化パイプラインでホストすることができます。

この方法で Java アプリケーションを配置する理由として、以下のいずれかが考えられます。

- Axis2 ハンドラー・インターフェースを使用する既存の投資がある。
- CICS パイプライン構成を使用したい。

注: Axis2 スタイルのアプリケーションは WEBSERVICE リソースを使用しません。それらは Axis2 プログラミング・モデルを使用して CICS と対話するため、CICS Web サービスのサポートの一部を使用できません。 Axis2 スタイルのアプリケーションでは、以下のサービスは完全にはサポートされません。

- SOAPFAULT CREATE
- WSACONTEXT GET
- DFHWS-OPERATION コンテナー
- DFHWS-MEP コンテナー
- DFHWS-USERID コンテナー
- DFHWS-TRANID コンテナー
- SOAP メッセージを保護するためのオプション

始める前に

JAX-WS を使用した POJO アプリケーションなど、Axis2 での配置に適した Java アプリケーションが必要です。 このタスクに関して、以下の POJO アプリケーションが例として使用されます。

```
/**
    * Simple example
    */
    @javax.jws.WebService(targetNamespace = "com.ibm.cics.example", name = "pojoExample")
    public class TestAxis2
    {
    public String getMessage(String input)
    {
    return "CICS got this: '" + input + "'";
    }
    }
```

このアプリケーションは、WSDL を生成するのに使用される XML 名前空間、および Web サービスに関連付ける名前を指定します。

このアプリケーションを TestAxis2.jar という jar ファイルにパッケージするため、このアプリケーションの Java コードをコンパイルし、JAX-WS 生成プログラムを実行する必要があります。以下のコードを実行することにより、これを行えます。

```
javac TestAxis2.java
wsgen -cp . TestAxis2 -wsdl
jar -cvf TestAxis2.jar *
```

JAX-WS 生成プログラムは、Axis2 によって使用される WSDL 文書およびバインディングも作成します。

このタスクについて

Axis2 Web サービスを配置するために、Web サービスのパイプライン・インフラストラクチャーを作成する必要があります。 パイプラインを作成したら、Web サ

ービスを作成することができます。 作成したパイプラインは、必要に応じた数の Web サービスで再利用できます。 以下のステップでは、パイプラインおよび Web サービスを作成する方法について説明しています。

注: このタスクでは WEBSERVICE リソースは作成されず、インストールもされません。

手順

1. パイプライン・インフラストラクチャーを作成します。
   a. Java パイプラインの Web サービス・インフラストラクチャーを作成します。 詳しくは、SOAP 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を参照してください。
   b. Axis2 リポジトリーを作成します。 そのためには、`$CICS_HOME/lib/pipeline/repository` に用意されているリポジトリーのコピーを作成します。
   c. パイプライン構成ファイルに `<repository>` エレメントを追加します。 このエレメントは、作成した Axis2 リポジトリーの名前を指定する必要があります。
   d. PIPELINE リソースを作成して使用可能にします。
2. パイプラインに関連付けられている Web サービスごとに、以下のステップを繰り返して、Web サービスを作成します。
   a. Axis2 アプリケーションを Axis2 リポジトリーに配置します。例えば、この例で作成した jar ファイルは、リポジトリー・ディレクトリーの `servicejars` というディレクトリーに配置する必要があります。 このディレクトリーが存在しない場合、作成しなければなりません。
   b. Web サービスの URIMAP リソースを定義し、インストールします。 URIMAP リソースでは、Web サービスに関連付けられた URI および PIPELINE リソースを指定する必要があります。 URI では、URI の Axis2 命名規則に従う必要があります。 デフォルトの Axis2 命名規則は、*/name_of_service*Service.*name_of_port*Port/*suffix* です。ここで、*name_of_service* は WSDL の Web サービスの名前、*name_of_port* は WSDL のポートの名前、*suffix* は自分で定義できるオプションの接尾部です。 前述の例の場合、以下の URIMAP リソースを使用できます。

      ```
      Urimap : EXAMPLE
      Group : EXAMPLE
      STatus : Enabled
      USAge : Pipeline
      SCheme : HTTP
      POrt : No
      HOST : *
      PAth : /TestAxis2Service.pojoExamplePort/example/TestAxis2
      TRansaction : CPIH
      PIpeline : EXAMPLE
      ```

      この例では、使用される PIPELINE リソースの名前を EXAMPLE と想定しています。

次のタスク

Web サービスが正常に実行されるかどうかをテストします。

XML を生成および解析する Java Web サービスの作成

XML 自体を解析および生成する Java アプリケーションを作成できます。 これらのアプリケーションは、他のプログラミング言語で記述された XML 認識アプリケーションとの整合性を持ちますが、XML を処理する上で、標準 Java テクノロジーを使用することによる利点があります。

手順

1. XML-ONLY WEBSERVICE リソースを作成します。 詳しくは、 643 ページの『XML 認識サービス・リクエスター・アプリケーションの作成』または 641 ページの『XML 認識サービス・プロバイダー・アプリケーション』を参照してください。
2. SOAP メッセージの本文の XML を生成して解析する Java Web サービスを作成します。 Java 6 Java Architecture for XML Binding (JAXB) ライブラリーなどの、さまざまなツールを使用して、これらの機能を備えた Java Web サービスの作成に役立てることができます。
3. オプション: プロバイダー・パイプラインを使用しており、SOAP 障害メッセージをリクエスターに戻す機能を追加する場合、JCICS SoapFault クラスを使用して、**EXEC CICS SOAPFAULT CREATE** コマンドを発行します。
4. オプション: リクエスター・パイプラインを使用している場合、JCICS サービス・クラスを使用し、**EXEC CICS INVOKE SERVICE** コマンドを発行します。

COBOL インターフェースがある Java Web サービスの作成

他のプログラミング言語の場合と同様の技法を使用して、CICS と対話する Java アプリケーションを作成できます。 これらのアプリケーションを作成するには、構造化された COMMAREA スタイルまたはコンテナー・スタイルのデータを作成する Java コードを作成または生成する必要があります。

手順

1. DFHWS2LS を使用して、Web サービスの COBOL 言語構造を作成します。
2. COBOL 言語構造を生成して解析する、Java Web サービスを作成します。 Java プログラムが既存の CICS アプリケーション・データへアクセスできるようにするツール、および COBOL 言語構造を生成して解析できる Java Web サービスの作成例のリンクについて詳しくは、Java からの構造化データとの対話を参照してください。
3. オプション: プロバイダー・パイプラインを使用しており、SOAP 障害メッセージをリクエスターに戻す機能を追加する場合、JCICS SoapFault クラスを使用して、**EXEC CICS SOAPFAULT CREATE** コマンドを発行します。
4. オプション: リクエスター・パイプラインを使用している場合、CICS SERVICE API とやり取りする JCICS サービス・クラスを使用し、**EXEC CICS INVOKE SERVICE** コマンドを発行します。

リクエスター・モードの JAX-WS Web サービスの配置

CICS で、JAX-WS アプリケーションをリクエスター・モード Web サービスとして配置できます。ただし、それらのアプリケーションは **EXEC CICS INVOKE** コマンドを使用しません。代わりに、JAX-WS を使用してリモート Web サービスと対話します。

始める前に

OSGi をサポートするよう JVM サーバーを構成する必要があります。詳しくは、Setting up a JVM serverを参照してください。

このタスクについて

リクエスター・モード Web サービスとして JAX-WS アプリケーションを配置することの利点は、zEnterprise Application Assist Processor (zAAP) を使用する、プラットフォーム非依存の Web サービス・リクエスター・アプリケーションが作成されるという点です。 zAAP を使用することにより、トランザクションのコストを削減できます。詳しくは、IBM Redbooks® 資料「zSeries Application Assist Processor (zAAP) Implementation」を参照してください。

手順

1. Java で Web サービス・リクエスター・アプリケーションを作成し、Java API for XML Web Services (JAX-WS) などの適切な API を使用して、リモート Web サービスを呼び出します。
2. オプション: JAX-WS を使用してリモート Web サービスを開始する場合、SOAP メッセージの生成、ネットワーク通信の処理、および SOAP 応答の処理のために JAX-WS も使用する必要があります。
3. Java アプリケーションを配置し、JVM サーバーにインストールします。

次のタスク

Web サービスが正常に開始されるかどうかをテストします。

Liberty JVM サーバーでの Java プロバイダー・モードの Web サービスの配置

Liberty JVM サーバーで、Web アプリケーションをプロバイダー・モード Web サービスとして配置できます。これらのアプリケーションは、Java 標準 JAX-WS および JAXB を使用して作成されます。このトピックは、CICS 統合モードの Liberty にのみ適用されます。

このタスクについて

CICS TS V5.3 には、最新の WebSphere Application Server Liberty Profile (WLP) が組み込まれています。この WLP によって、Java API for XML Web Services (JAX-WS) および Java Architecture for XML Binding (JAXB) 用の機能が提供されます。また、これらのテクノロジーを使用すると、SOAP Web サービスを CICS アプリケーションの一部として Java で作成できます。以下の記事では、Eclipse のセットアップ方法、サンプル Web サービス・プロジェクトのテスト方法、サンプルを CICS に配置する方法、JCICS を使用するようにサンプルを変更する方法、およびそれを Web サービス・エクスプローラーでテストする方法を示します。詳しくは、CICS DevCenter の記事「JAX-WS web service sample for Liberty」を参照してください。

この方法で Java アプリケーションを配置する理由として、以下のいずれかが考えられます。

- Java で Web サービスを作成する。
- CICS Web サービス・アシスタントを使用して扱うことが困難な、複雑な WSDL 文書がある。
- Web サービス・アプリケーションの処理を zEnterprise Application Assist Processor (zAAP) にオフロードする。

注: Liberty JVM サーバーに配置された Web アプリケーションは、WEBSERVICE リソースおよび TCPIPSERVICE リソースを使用しません。それらは Liberty HTTP リスナーを使用して Web 要求と対話するため、CICS Web サービスのサポートの機能を使用できません。

SOAP メッセージの検証

CICS Web サービスを使用するときには、Web サービス記述に含まれるスキーマに SOAP メッセージが準拠しているかどうか、検証するよう指定できます。プロバイダーとリクエスターの両方のモードのアプリケーションを検証できます。

始める前に

Web サービス配置の開発中およびテスト時に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の問題を検出するうえで役立ちます。 ただし、SOAP メッセージの完全な検証によってかなりのオーバーヘッドが生じるため、完全にテストされる実動アプリケーションでメッセージを検証することはお勧めできません。

CICS 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、SOAP メッセージを検証するために、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。

このタスクについて

SOAP メッセージは、アプリケーション・データ構造に変換される前、およびアプリケーション・データ構造から SOAP メッセージが生成されるときに検証されます。 SOAP メッセージは WSDL で XML スキーマを使用して検証され、CICS の変換要件に照らして再び検証されます。 WEBSERVICE リソースの**WSDLFILE**属性に指定されている WSDL ファイルか、WEBSERVICE リソースの**ARCHIVEFILE** 属性に指定されている `.zip`ファイルに格納された WSDL ファイルを使用できます。もし両方の属性が指定されると、**ARCHIVEFILE** 属性で指定されたアーカイブ・ファイルの WSDL ファイルが使用されます。

検証をオフにすると、CICS は Java プログラムを使用しません。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。したがって、SOAP メッセージは正常に検証できますが、ランタイム環境では失敗する可能性があり、またその逆も同様です。

手順

1. CICS 領域内に OSGi JVM サーバーをセットアップします。DFHPIVAL を使用した SOAP 検証は、OSGi フレームワーク内でのみ実行され、Axis2 または Liberty プロファイル JVM では実行されません。

a. グループ DFH$OSGI にサンプル JVM サーバー DFH$JVMS をインストールするか、独自の JVM サーバーを作成します。 詳しくは、Setting up a JVM serverを参照してください。
b. 独自の JVM サーバーを作成した場合は、グループ DFHPIVAL 内の DFHPIVAL プログラム定義を変更して、JVMSERVER リソースの名前を参照するようにします。 DFHPIVAL 定義はロックされておらず、編集可能です。 デフォルトでは、この定義は DFH$JVMS を参照しています。

2. 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。 この関連は WEBSERVICE リソース用に作成され、パイプライン走査中にパイプラインのピックアップ・ディレクトリーに 1 つ以上の WSDL ファイルを含む WSDL ファイルまたは `.zip`ファイルが見つかった際に、自動的に作成される WEBSERVICE リソース用に作成されます。

   RDO で作成される WEBSERVICE 定義の場合、Web サービス記述は WSDLFILE 属性で指定されます。

3. WEBSERVICE リソースの **`VALIDATION=YES`** 属性を指定して、Web サービスの検証をオンにします。 リソースを定義するときに検証が必要かどうかを指定できます。この設定は、リソースのインストール後に変更することもできます。

タスクの結果

システム・ログを確認して、SOAP メッセージが有効かどうか調べます。メッセージ DFHPI1002 は、SOAP メッセージが正常に検証されたことを示し、メッセージ DFHPI1001 は、検証が失敗したことを示します。

次のタスク

不要になったら、検証をオフにします。

アプリケーション提供の無効および初期化されていないデータの処理

JSON とアプリケーション間のデータの変換中に検出された無効なデータを許容する方法を説明します。

既存のアプリケーションを有効にするサービスにより、CICS は、そのアプリケーションで使用されている COMMAREA またはコンテナーの内容を認識するようになります。CICS にデプロイされる WSBind ファイルには、個々のフィールドの名前、場所、データ型など、データ形式に関する情報が含まれています。 CICS はこの情報を使用して、XML/JSON データ表記とアプリケーション・データとの間の変換を容易にします。

アプリケーション・データが WSBind ファイルに保管されている情報と整合していない場合、CICS は問題を報告し、データの変換に失敗します。例えば、アプリケーション・インターフェースは変更されますが、WSBind ファイルは再生成も再デプロイもされません。このような問題が DFHPI1010 エラー・メッセージの原因となることはよくあります。

アプリケーションがフィールドを意図的に初期化しないまま残すというシナリオでは、この状態のバリアントとして、より捉えにくい問題が発生する場合があります。例えば、符号付きパック 10 進数フィールドに小さい値 (ヌル・バイト) が格納

されているとします。CICS はそのフィールドを処理する際に、この無効な値が意図的に初期化されていないことを判断できません。そのため、CICS は DFHPI1010 エラー・メッセージを表示し、データ変換は失敗します。アプリケーションがフィールドを初期化しないまま残す理由はさまざまです。以下に例を示します。

- 言語構造では入力データ形式と出力データ形式を記述できるが、初期化されていないフィールドは入力専用である場合。
- アプリケーションがエラーを検出し、応答コードを設定した後、他の出力フィールドを初期化せずに戻った場合。
- アプリケーション内の条件付きロジックでフィールドを有意であると見なすかどうかを制御する場合。

このシナリオに対する理想的な対応は、無効な値が処理対象として CICS に渡されることがないようにアプリケーションを変更することです。あるいは、代わりの方法として、アシスタントの `DATA-SCREENING` オプションを WSBind ファイルの生成時に `DISABLED` に設定することもできます。このオプションにより、CICS はアプリケーション提供の不正なデータを許容し、そのようなデータをデフォルト値 (通常はゼロ) で置き換えます。このようなオプションは、既存のアプリケーションのサービス使用可能化を単純にしますが、エラーの検出を困難にすることから、慎重に使用する必要があります。データ・スクリーニングが無効にされていると、アプリケーションによって生成されたデータのエラーが CICS で検出されない可能性が高くなります。

アプリケーション生成の配列に部分的にデータが取り込まれないままになる場合にも、これに関連する状態が発生する可能性があります。例えば、1000 個のデータ・レコードからなる配列があり、アプリケーションが最初の 10 個のレコードを初期化するとします。CICS がデータの XML 表記または JSON 表記を生成して 1000 個すべてのフィールドにデータを取り込むと、そのうち 990 個のフィールドが空になってしまいます。この問題に対する理想的な対応は、データを取り込む予定のレコード数を正確に指定する `OCCURS DEPENDING ON` 節をアプリケーションに導入することです。あるいは、代わりの方法として、WSBind ファイルの生成時にアシスタントの `TRUNCATE-NULL-ARRAYS` オプションを使用することもできます。このオプションは CICS に対し、初期化されていないデータを検出し、そのポイントで配列を切り捨てるように指示します。このオプションを使用すると、未確定の初期化されていないアプリケーション・データから生成される JSON/XML データが簡潔になりますが、配列に含まれるデータが初期化されていないデータと区別できない場合、誤ってデータが損失するリスクがもたらされます。

最善のソリューションは、曖昧ではないデータをアプリケーションに生成させ、そのデータを記述する言語構造と一貫して整合性を持たせるようにすることです。`TRUNCATE-NULL-ARRAYS=ENABLED` および `DATA-SCREENING=DISABLED` オプションを使用すると、CICS がアプリケーション提供の不完全なデータを許容するようになりますが、プロセスに危険な要素および不確実性がもたらされることにもなります。

以下の例で、これらのオプションの使用方法を示します。

例 1: 10 進数フィールドの耐障害性

10 進数フィールドの不正なデータの自動修正をテストします。

このタスクについて

このシナリオでは、不正なデータが含まれる 10 進数フィールドに自動的にデフォルト値 0 を設定する方法を示します。

手順

1. JSON スキーマと必要な成果物を生成し、CICS が JSON と COBOL アプリケーション・データとの間で変換を行えるようにします。
   a. コピーブックを準備します。

```
03 BAD-DATA.
05 NORMAL-NUM PIC 9(2).
05 NORMAL-CHAR PIC X(3).
05 PZONED-DECIMAL PIC S9(4) DISPLAY.
05 NZONED-DECIMAL PIC S9(4) DISPLAY.
05 UZONED-DECIMAL PIC 9(4) DISPLAY.
05 NORMAL-CHAR2 PIC X(3).
05 PBINARY PIC S9(4) BINARY.
05 NBINARY PIC S9(4) COMP.
05 UBINARY PIC 9(4) COMP.
05 NORMAL-NUM2 PIC 9(3).
05 PPACKED-DECIMAL PIC S9(4) COMP-3.
05 NPACKED-DECIMAL PIC S9(4) COMP-3.
05 UPACKED-DECIMAL PIC 9(4) COMP-3.
05 NORMAL-NUM3 PIC 9(2).
05 NORMAL-CHAR3 PIC X(3).
05 FLOAT-ZONED PIC S9(4)V99.
05 FLOAT-PACKED PIC S9(4)V99 COMP-3.
```

   b. JCL を実行依頼します。 この機能をアクティブにするために、DATA-SCREENING を DISABLED に設定します。

```
//GENJSON JOB ('accounting information',name),CLASS=M,REGION=0M,
// MSGCLASS=A,NOTIFY=&SYSUID
//JCLLIB JCLLIB ORDER=ZZZZ.A.ZOSCONN.JCL
//LS2JS EXEC DFHLS2JS,
// JAVADIR='/java/java7_64/J7.0_64',
// USSDIR='cics.ts.test/tolerate',
// PATHPREF='',
// TMPDIR='/tmp',
// TMPFILE=''
//INPUT.SYSUT1 DD *
PDSLIB=ZZZZ.A.ZOSCONN.COPYBOOK
REQMEM=BADDATA
RESPMEM=BADDATA
JSON-SCHEMA-REQUEST=/u/zzzz/json/ls2js/baddata1_request.json
JSON-SCHEMA-RESPONSE=/u/zzzz/json/ls2js/baddata1_response.json
LANG=COBOL
LOGFILE=/u/zzzz/json/ls2js/baddata1.log
MAPPING-LEVEL=4.1
DATA-SCREENING=DISABLED
CHAR-VARYING=COLLAPSE
PGMNAME=baddata1
URI=/baddata1
PGMINT=COMMAREA
WSBIND=/u/zzzz/json/ls2js/baddata1.wsbind
/*
```

   c. 生成された JSON スキーマ (要求と応答) と WSBind ファイルが正常に作成されていることを確認します。
2. アプリケーション・プログラムを作成します。
   a. COBOL プログラム内で、10 進数フィールド以外のすべてのフィールドに値を割り当てます。

```
MOVE LOW-VALUE TO BAD-DATA.
MOVE 11 TO NORMAL-NUM.
MOVE 'AAA' TO NORMAL-CHAR.
MOVE 'BBB' TO NORMAL-CHAR2.
MOVE 222 TO NORMAL-NUM2.
MOVE 'CCC' TO NORMAL-CHAR3.
```

3. CICS リソースを定義します。
   a. COBOL プログラムを定義してインストールします。
   b. TCPIPSERVICE プログラムを定義してインストールします。
   c. ご使用のパーサーに応じて、PIPELINE を定義してインストールします。
4. アプリケーションをテストします。
   a. PIPELINE SCAN を実行します。WSbind ファイルに記述されている Web サービスには、INSERVICE を使用します。
   b. JSON 要求を送信します。

```
{"BADDATA1Operation":{"bad_data":{"normal_num":12}}}
```

   c. 応答を確認します。

タスクの結果

初期化されていないフィールドには、デフォルト値が割り当てられています。

```
{
"BADDATA1OperationResponse": {
"bad_data": {
"normal_num": 11,
"normal_char": "AAA",
"pzoned_decimal": 0,
"nzoned_decima": 0,
"uzoned_decimal": 0,
"normal_char2": "BBB",
"pbinary": 0,
"nbinary": 0,
"ubinary": 0,
"normal_num2": 222,
"ppacked_decimal": 0,
"npacked_decimal": 0,
"upacked_decimal": 0,
"normal_num3": 0,
"normal_char3": "CCC",
"float_zoned": 0,
"float_packed": 0
}
}
}
```

第 4 章 Web サービスを保護するためのサポート

CICS Transaction Server for z/OS では、SOAP および JSON メッセージを保護するために使用できる多数の関連テクノロジーがサポートされます。

これらのテクノロジーのいくつかは HTTP プロトコルに含まれており、SOAP と JSON の両方に等しく適用されます。 Web Services Security (WSS): SOAP Message Security 1.0 仕様を使用する、SOAP でのみ使用可能なものもあります。共用される TCP/IP および HTTP セキュリティー・オプションについては、Security for TCP/IP clientsおよびCICS Web サポートのセキュリティーを参照してください。

SAML アサーションの使用については、SAML 用の CICS の構成を参照してください。

SOAP Web サービスのセキュリティー

Web Services Security (WSS): SOAP Message Security 1.0 は、SOAP メッセージを保護し認証するためのセキュリティー・トークン およびデジタル署名 の使用について記述しています。詳しくは、Web Services Security: SOAP Message Security 1.0 仕様を参照してください。 Web Services Security: SOAP Message Security 1.0

Web Services Security は、メッセージが不正に開示されたり、不正または気付かれずに変更されたりしないようにすることによって、SOAP メッセージのプライバシー と保全性 を保護します。 WSS は、メッセージ内の XML エレメントをデジタル署名および暗号化することでこのような保護を提供します。 保護できるエレメントは、本体または本体やヘッダー内のエレメントです。 SOAP メッセージ内の異なるエレメントに対して異なるレベルの保護を適用することができます。

Web Services Trust Language (WS-Trust) 仕様は、セキュリティー・トークンを要求して発行するためのフレームワークを提供し、Web サービス・リクエスターと Web サービス・プロバイダー間の信頼関係を管理することによって、Web Services Security をさらに拡張します。 SOAP メッセージの認証をこのように拡張することによって、Web サービスは、信頼のおける第三者機関を使用して、さまざまなタイプのセキュリティー・トークンを検証および交換することができます。 この第三者機関は、*Security Token Service* (STS) と呼ばれます。 Web Services Trust Language の詳細については、Web Services Trust Language 仕様を参照してください。

CICS Transaction Server for z/OS では、パイプライン内で CICS 提供のセキュリティー・ハンドラーを使用することにより、これらの仕様がサポートされます。

- アウトバウンド・メッセージでは、CICS は SOAP 本体全体にデジタル署名して暗号化するためのサポートを提供します。また CICS は、STS を使用して、さまざまなタイプのセキュリティー・トークンでユーザー名トークンを交換することができます。

© Copyright IBM Corp. 2005, 2018

- インバウンド・メッセージでは、CICS は、本体または本体やヘッダーのエレメントが暗号化されているかデジタル署名されているメッセージをサポートします。また CICS は、STS を使用してセキュリティー・トークンを交換して検証することもできます。

CICS は、個別の Trust クライアント・インターフェースを提供して、CICS セキュリティー・ハンドラーを使用せずに STS とデータをやり取りすることもできます。

注**:** Web Services Security は SP800-131A に準拠しない可能性があります。 Web Services Security はパイプライン内にハンドラーを追加することで構成され、CICS はカスタム作成ハンドラーでの処理の制御を行うことができません。デジタル署名を使用する場合、アルゴリズム `dsa-sha1` および `rsa-sha1` だけを指定できます。これらのアルゴリズムは SP800-131A に準拠していません。 SOAP 本体の暗号化に使用できる、2 つの鍵を使った Triple-DES 暗号化アルゴリズムもまた準拠していません。

Web Services Security を実装するための前提条件

Web Services Security を実装するには、次のような更新を CICS 領域に適用する必要があります。IBM XML Toolkit for z/OS v1.10 をインストールし、APAR OA14956 を適用し、3 つのライブラリーを DFHRPL 連結に追加します。

このタスクについて

Web Services Security を実装する前に、以下の手順を完了してください。

手順

1. 無料の IBM XML Toolkit for z/OS v1.10 をインストールします。 これは、サイト http://www.ibm.com/servers/eserver/zseries/software/xml/ からダウンロードできます。 バージョン 1.10 をインストールする必要があります。それより後のバージョンは、CICS の Web Services Security サポートでは機能しません。
2. ICSF APAR OA14956 が z/OS にまだインストールされていない場合は、これを適用します。
3. 次のライブラリーを DFHRPL 連結に追加します。
   - *hlq*.SIXMLOD1。*hlq* は、XML ツールキットの高位修飾子です。
   - *hlq*.SCEERUN。*hlq* は、言語環境の高位修飾子です。
   - *hlq*.SDFHWSLD。*hlq* は、 CICS インストール済み環境の高位修飾子 (例えば、CICSTS55) です。

   最初の 2 つのライブラリーには、実行時にセキュリティー・ハンドラーで必要な DLL が含まれています。 IXM4C57 は XML ツールキットによって提供され、*hlq*.SIXMLOD1 にあります。C128N は言語環境ランタイムによって提供され、*hlq*.SCEERUN にあります。

   hlq.SDFHWSLD ライブラリーを使用すると、CICS は DFHWSSE1 と DFHWSXXX の Web Services Security モジュールを検索できるようになります。

4. **EDSALIM** システム初期化パラメーターの値を増やさなければならないことがあります。 ロードされる 3 つの DLL は、約 15 MB の EDSA ストレージを必要とします。

タスクの結果

ライブラリーを指定していない場合は、次のメッセージが表示されます。

```
CEE3501S The module module_name was not found.
(CEE3501S モジュール module_name が見つかりませんでした。)
```

module_name は、欠落しているライブラリーによって異なります。

SOAP Web サービスの保護の計画

Web サービスを保護するための最良の方法を判別します。 CICS は、構成可能なセキュリティー・メッセージ・ハンドラーや、個別の Trust クライアント・インターフェースなど、多くのオプションをサポートしています。

このタスクについて

CICS は、Web サービスごとではなくパイプライン・レベルで、Web Services Security (WS-Security または WSS) を実装します。 以下の質問に答えて、セキュリティーを実装する最良の方法を判別してください。

手順

1. パイプライン処理のパフォーマンスは重要ですか? WSS を使って Web サービスを保護する場合、パフォーマンスがかなり影響を受けます。

   WSS を実装する主な利点は、SOAP メッセージの一部を暗号化することによって、一連の中間ノードを介してメッセージを送信できることです。これらの中間ノードはすべて、ルーティングまたは処理の決定を行うために SOAP ヘッダーを調べることができますが、メッセージの内容を表示することはできません。機密にすべきセクションだけを暗号化することには、以下のような利点があります。

   - 一連の中間プロセス内のすべてのノードで暗号化および暗号化解除が行われることによるオーバーヘッドが生じません。
   - データの最終的な受信側からの理解を得られる場合に限り、信頼できないノードの公衆網を介して機密メッセージを送付することができます。

   WSS の使用に代わる方法として、SSL を使用してデータ・ストリーム全体を暗号化することができます。
2. WSS を使用する場合、どのレベルのセキュリティーが必要ですか? このオプションは、メッセージ・ヘッダーがユーザー名およびパスワードを含む基本認証から、メッセージでのデジタル署名と暗号化の組み合わせまで、多岐にわたります。 CICS セキュリティー・ハンドラーがサポートするオプションについては、 658 ページの『SOAP メッセージを保護するためのオプション』で説明しています。
3. CICS 提供のセキュリティー・ハンドラーは、要件を満たしていますか? より高度なセキュリティー処理を実行する場合は、独自にカスタムのセキュリティー・

ハンドラーを作成する必要があります。 このハンドラーは、RACF によって直接か、または Security Token Service を使用して、メッセージの必要な認証を実行し、デジタル証明書および暗号化されたエレメントの処理を行う必要があります。 詳しくは、 675 ページの『カスタムのセキュリティー・ハンドラーの作成』を参照してください。

4. パイプラインに MTOM ハンドラーが含まれていますか? パイプライン構成ファイルで、MTOM ハンドラーとセキュリティー・ハンドラーの両方を使用できるようにする計画の場合、MIME Multipart または Related メッセージはすべて、互換モードで処理されるため、セキュリティー・ハンドラーはメッセージ本文の XOP エレメントを構文解析できません。 この処理は、パイプライン処理のパフォーマンスに、さらに影響を与える恐れがあります。

SOAP メッセージを保護するためのオプション

CICS では、SOAP メッセージの署名と暗号化の両方がサポートされるため、SOAP メッセージで送受信するデータに最適なセキュリティー・レベルを選択することができます。

プロバイダー・モードの Axis2 Web サービス Java アプリケーション、または Axis2 の MessageContext を使用してパイプラインに接続するプロバイダー Web サービスでは、SOAP メッセージの署名および暗号化はサポートされていません。

以下のオプションの中から選択できます。

トラステッド認証
: サービス・プロバイダー・パイプラインでは、CICS は、SOAP メッセージ・ヘッダーのユーザー名トークンを、信頼できるものとして受け入れることができます。 この通常ユーザー名とパスワードを含むタイプのセキュリティー・トークンですが、この場合、パスワードは不要です。 CICS は提供されたユーザー名を信頼し、それを DFHWS-USERID コンテナーに置きます。メッセージはパイプラインで処理されます。

  サービス・リクエスターのパイプラインでは、CICS は、SOAP メッセージ・ヘッダーにパスワードがないユーザー名トークンを、サービス・プロバイダーに送信することができます。

基本認証
: サービス・プロバイダー・モードでは、CICS は、インバウンド SOAP メッセージでの認証のために、SOAP メッセージ・ヘッダーのユーザー名トークンを受け入れることができます。このユーザー名とパスワードを含むタイプのセキュリティー・トークンです。 CICS は、RACF などの外部セキュリティー・マネージャーを使用して、ユーザー名トークンを検証します。成功すると、ユーザー名はコンテナー DFHWS-USERID に置かれ、SOAP メッセージがパイプラインで処理されます。 CICS がユーザー名トークンを検証できない場合は、SOAP 障害メッセージがサービス・リクエスターに戻されます。

  パスワードを含むユーザー名トークンは、サービス・リクエスター・モードや、アウトバウンド SOAP メッセージではサポートされません。

HTTP 基本認証
: サービス・プロバイダー・モードでは、CICS は、HTTP プロトコルでの基

本認証情報を受け入れることができます。 サービス・リクエスターは URIMAP 定義を使用して資格情報 (ユーザー識別情報) がグローバル・ユーザー出口 XWBAUTH によって収集されることを指定します。 XWBAUTH は要求に応じてこの情報を CICS に受け渡し、CICS は HTTP 許可ヘッダーの情報をサービス・プロバイダーに送信します。

拡張認証

サービス・プロバイダーおよびリクエスター・パイプラインでは、認証の目的で、Security Token Service (STS) によってセキュリティー・トークンを検証または交換できます。 この認証により、CICS は、メッセージ・ヘッダーにセキュリティー・トークンのある、通常はサポートされないメッセージ (Kerberos トークンや SAML アサーションなど) の送受信を行うことができるようになります。

インバウンド・メッセージの場合は、セキュリティー・トークンの検証または交換を選択できます。 要求が、セキュリティー・トークンの交換である場合、CICS は、STS からユーザー名トークンを受け取る必要があります。 アウトバウンド・メッセージの場合は、セキュリティー・トークンに関するユーザー名トークンの交換だけが可能です。

X.509 証明書による署名

サービス・プロバイダー・モードとサービス・リクエスター・モードでは、SOAP メッセージ・ヘッダーで X.509 証明書を提供して、認証のために SOAP メッセージの本文に署名することができます。 このバイナリー・セキュリティー・トークン として知られるタイプのセキュリティー・トークンです。 インバウンド SOAP メッセージからのバイナリー・セキュリティー・トークンを受け入れるには、証明書に関連付けられた公開鍵を RACF などの外部セキュリティー・マネージャーにインポートして、**KEYRING** システム初期化パラメーターで指定された鍵リングに関連付ける必要があります。アウトバウンド SOAP メッセージでは、公開鍵を生成して、意図した受信側に発行されます。 公開鍵の生成には、Integrated Cryptographic Service Facility (ICSF) が使用されます。

X.509 デジタル証明書に関連付けられたラベルを指定する場合は、次の文字は使用しないでください。

```
< > : ! =
```

また、ヘッダーに 2 番目の X.509 証明書を含めて、最初の証明書を使用して署名することができます。 この 2 番目の証明書により、2 番目の X.509 証明書に関連付けられたユーザー ID を使用して CICS で作業を実行できるようになります。 SOAP メッセージに署名するために使用する証明書は、トラステッド・ユーザー ID に関連付けられている必要があります。また、異なる ID (宣言 *ID*) に関連付けられたパスワードをトラステッド・ユーザー ID が持たなくても、この ID で作業を実行することを表明するために代理権限も必要です。

暗号化

サービス・プロバイダー・モードおよびサービス・リクエスター・モードでは、Triple-DES や AES などの対称アルゴリズムを使用して、SOAP メッセージの本文を暗号化することができます。 対称アルゴリズムでは、データの暗号化と暗号化解除に同じ鍵が使用されます。 この鍵は、対称鍵 として知られています。この鍵はメッセージに含められ、意図した受信側の公開

鍵と非対称鍵暗号化アルゴリズム RSA 1.5 との組み合わせを使用して暗号化されます。 非対称アルゴリズムは複雑で、対称鍵を暗号化解除するのは困難であるため、この暗号化によってセキュリティーが強化されます。 また一方、SOAP メッセージの大部分は、より迅速に暗号化解除できる対称アルゴリズムで暗号化されるため、パフォーマンスが向上します。

インバウンド SOAP メッセージでは、SOAP 本体のエレメントを暗号化してから、SOAP 本体を全体として暗号化することができます。 暗号化の種類は特に、機密データを含むエレメントに適していることがあります。 CICS が 2 つのレベルで暗号化された SOAP メッセージを受信すると、CICS は両方のレベルを自動的に暗号化解除します。暗号化の種類は、アウトバウンド SOAP メッセージではサポートされていません。

CICS では、メッセージ・ヘッダーのみに暗号化されたエレメントを含み、SOAP 本体のエレメントは暗号化されていないインバウンド SOAP メッセージはサポートされません。

署名および暗号化
: サービス・プロバイダー・モードおよびサービス・リクエスター・モードでは、SOAP メッセージの署名と暗号化の両方を選択することができます。 CICS は必ず、最初に SOAP メッセージの本文に署名してから、暗号化します。 この方法の利点は、メッセージの機密性と保全性の両方を確保できる点です。

ICRX ベースの **ID** 伝搬
: サービス・プロバイダー・モードでは、非認証 WS-Security ユーザー ID トークンを使用する場合と同じ状況で、非認証 ICRX (Extended Identity Context Reference) ID トークンを使用できます。 ICRX ID トークンは、ユーザー ID へマップされる z/OS ID です。 CICS は ICRX ID トークンをユーザー ID に解決し、DFHWS-ICRX コンテナーにコピーを置きます。 また、CICS は DFHWS-USERID コンテナーにデータを入れます。 ICRX ID トークンについて詳しくは、Identity propagation and distributed securityを参照してください。

Security Token Service を使用した認証

CICS は、Tivoli Federated Identity Manager などの Security Token Service (STS) と相互運用して、Web サービスのより高度な認証を提供することができます。

STS は、信頼のおける第三者機関として働き、Web サービス・リクエスターと Web サービス・プロバイダー間の信頼関係を仲介する Web サービスです。 SSL ハンドシェークでの認証局と同様の方法で、STS は、メッセージが示す資格情報をリクエスターとプロバイダーが「信頼」できることを保証します。 この信頼関係は、セキュリティー・トークンの交換によって表されます。 STS は、これらのセキュリティー・トークンを発行、交換、および検証して信頼関係を確立し、さまざまな信頼ドメインからの Web サービス同士が正常に対話できるようにします。詳しくは、Web Services Trust Language の説明を参照してください。

CICS は Trust クライアントとして働き、2 つのタイプの Web サービス要求を STS に送信できます。 要求の 1 つ目のタイプは、WS-Security メッセージ・ヘッ

ダーのセキュリティー・トークンを検証することであり、要求の 2 つ目のタイプは、セキュリティー・トークンを別のタイプに交換することです。 これらの要求により、多岐にわたる信頼ドメインからのさまざまなセキュリティー・トークン (SAML アサーションや Kerberos トークンなど) を含むメッセージを、CICS で送受信できるようになります。

CICS セキュリティー・ハンドラーを構成して、CICS が STS とデータをやり取りする方法を定義するか、独自のメッセージ・ハンドラーを作成して、個別に提供される Trust クライアント・インターフェースを使用することができます。 選択したメソッドに関わらず、SSL を使用して CICS と STS の間の接続を保護します。

セキュリティー・ハンドラーで STS を呼び出す方法

CICS セキュリティー・ハンドラーは、パイプライン構成ファイルの情報を使用して、Web サービス要求を Security Token Service (STS) に送信します。 送信される要求のタイプは、STS で実行するアクションによって異なります。

サービス・プロバイダー・パイプラインの場合
: サービス・プロバイダー・パイプラインでは、セキュリティー・ハンドラーの構成方法に応じて、次のような 2 種類のアクションがセキュリティー・ハンドラーによってサポートされます。

  - インバウンド・メッセージの WS-Security ヘッダーにある、セキュリティー・トークンの最初のインスタンスまたは特定タイプの最初のセキュリティー・トークンを検証するために、STS へ要求を送信します。
  - インバウンド・メッセージの WS-Security ヘッダーにある、セキュリティー・トークンの最初のインスタンスまたは特定タイプの最初のセキュリティー・トークンを、CICS が理解できるセキュリティー・トークンに交換するために、STS へ要求を送信します。

  セキュリティー・ハンドラーは、動的にパイプラインを作成し、Web サービス要求を STS に送信します。 このパイプラインは、STS からの応答が受信されるまで存在し、その後に削除されます。 要求が成功した場合、STS は、ID トークンまたはトークンの妥当性の状況を戻します。 セキュリティー・ハンドラーは、トークンから派生した RACF ID を DFHWS-USERID コンテナーに配置します。

  STS でエラーが発生した場合、STS は SOAP 障害をセキュリティー・ハンドラーに戻します。 その後セキュリティー・ハンドラーは、Web サービス・リクエスターに障害を戻します。

サービス・リクエスター・パイプラインの場合
: サービス・リクエスター・パイプラインでは、セキュリティー・ハンドラーが要求できるのは、STS によってトークンを交換することのみです。 パイプライン構成ファイルは、STS がセキュリティー・ハンドラーに対して発行する必要があるトークンのタイプを定義します。

  要求が成功した場合、RACF ID は DFHWS-USERID コンテナーに配置され、トークンはアウトバウンド・メッセージ・ヘッダーに組み込まれます。 STS でエラーが発生した場合、STS は SOAP 障害をセキュリティー・ハンドラーに戻します。 その後セキュリティー・ハンドラーは、パイプラインを介して、障害を Web サービス・リクエスター・アプリケーションに戻します。

セキュリティー・ハンドラーは、パイプラインに関する 1 つのタイプのアクションだけを STS に要求できます。 また、アウトバウンド要求メッセージに関する 1 つのタイプのトークンだけを交換でき、WS-Security メッセージ・ヘッダー内の最初のトークン (最初のインスタンス、または特定のタイプの最初のインスタンス) だけを限定的に処理します。 これらのオプションは、STS の使用に関する一般的なシナリオのほとんどをカバーしますが、インバウンドおよびアウトバウンド・メッセージを扱う際に必要となる処理を提供しない場合もあります。

より限定的な処理を準備してインバウンド・メッセージ・ヘッダーで多くのトークンを扱うようにする場合、または複数のタイプのトークンをアウトバウンド・メッセージと交換する場合は、Trust クライアント・インターフェースを使用します。このインターフェースを使用すると、カスタムのメッセージ・ハンドラーを作成して、独自の Web サービス要求を STS に送信できます。

Trust クライアント・インターフェース

Trust クライアント・インターフェースによって、セキュリティー・ハンドラーを使用せずに、直接 Security Token Service (STS) と対話することができます。 この方法を使用すると、セキュリティー・ハンドラーを使った処理よりも高度な処理をトークンに対して柔軟に実行できます。

Trust クライアント・インターフェースは、CICS 提供のプログラム DFHPIRT を拡張したものです。 このプログラムは通常、CICS Web サービス・アシスタントを使用して Web サービス・リクエスター・アプリケーションが配置されていない場合に、パイプラインを開始するために使用されます。ただし、STS に対する Trust クライアント・インターフェースとして機能することもできます。

Trust クライアント・インターフェースを起動するには、メッセージ・ハンドラーまたはヘッダー処理プログラムから DFHPIRT にリンクして、DFHWSTC-V1 と呼ばれるチャネルおよびセキュリティー・コンテナー一式を渡します。 これらのコンテナーを使用することで、柔軟性が高められ、STS の検証または実行アクションのいずれかを要求し、交換するトークンのタイプを選択し、メッセージ・ヘッダーの該当するトークンを渡すことができます。 DFHPIRT は動的にパイプラインを作成し、セキュリティー・コンテナーからの Web サービス要求を構成し、それを STS に送信します。

DFHPIRT は、STS から応答が戻るのを待機し、それを DFHWS-RESTOKEN コンテナーに入れてメッセージ・ハンドラーに渡します。 STS でエラーが発生した場合、STS は SOAP 障害を戻します。 DFHPIRT は、その障害を DFHWS-STSFAULT コンテナーに入れ、パイプライン内のリンクしているプログラムに戻します。

サービス・プロバイダーおよびサービス・リクエスター・パイプラインでセキュリティー・ハンドラーを使用できるようにしなくても、Trust クライアント・インターフェースを使用できます。あるいは、セキュリティー・ハンドラーに追加して、Trust クライアント・インターフェースを使用することもできます。

SOAP メッセージへの署名

インバウンド・メッセージでは、CICS は SOAP 本体内のエレメントおよび SOAP ヘッダー・ブロックのデジタル署名をサポートしています。 アウトバウンド・メッセージでは、CICS は SOAP 本体内のすべてのエレメントに署名します。

SOAP メッセージは、<Envelope> エレメントから構成される XML 文書です。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。

WSS: SOAP Message Security 仕様では、<Header> と <Body> の内容にエレメント・レベルで署名することができます。 つまり、あるメッセージでは、署名するエレメントと署名しないエレメントがあり、別の署名または別のアルゴリズムを使用して署名することができます。 例えば、オンライン購入アプリケーションで使用される SOAP メッセージでは、受注を確認するエレメントは法的状況を持つことがあるため、これらのエレメントには署名するのが適しています。 しかし、メッセージ全体への署名にかかるオーバーヘッドを避けるために、一部の情報は署名しないでおくことも可能であるかもしれません。

インバウンド・メッセージでは、セキュリティー・メッセージ・ハンドラーが、SOAP <Header> および <Body> 内の個々のエレメントのデジタル署名を検証することができます。

- <Header> で検出される署名済みエレメント。
- SOAP <Body> 内の署名済みエレメント。 署名済みの本体を受け入れるようハンドラーが構成されている場合、CICS は本体が署名されていない SOAP メッセージをすべて拒否して、SOAP 障害を発行します。

アウトバウンド・メッセージでは、セキュリティー・メッセージ・ハンドラーが署名できるのは、<Body> だけで、<Header> には署名しません。 アルゴリズム、および本体への署名に使用される鍵は、ハンドラーの構成情報で指定されます。

署名アルゴリズム

CICS は、XML Signature 仕様で必要な署名アルゴリズムをサポートします。 それぞれのアルゴリズムは、汎用リソース ID (URI) で識別されます。

アルゴリズム	**URI**
Digital Signature Algorithm と Secure Hash Algorithm 1 (DSA と SHA1) インバウンド SOAP メッセージでのみサポートされます。	http://www.w3.org/2000/09/xmldsig#dsa-sha1
Rivest-Shamir-Adleman アルゴリズムと Secure Hash Algorithm 1 (RSA と SHA1)	http://www.w3.org/2000/09/xmldsig#rsa-sha1

署名された SOAP メッセージの例

これは、CICS によって署名された SOAP メッセージを示した例です。

```
<?xml version="1.0" encoding="UTF8"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Header>
 <wsse:Security xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
                xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd"
                xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
                xmlns:xenc="http://www.w3.org/2001/04/xmlenc#" SOAP-ENV:mustUnderstand="1">
 <wsse:BinarySecurityToken 1
              EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#Base64Binary"
              ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509"
              wsu:Id="x509cert00">MIIChDCCAe2gAwIBAgIBADANBgkqhkiG9w0BAQUFADAwMQswCQYDVQQGEwJHQjEMMAoGA1UEChMD
                                  SUJNMRMwEQYDVQQDEwpXaWxsIFlhdGVzMB4XDTA2MDEzMTAwMDAwMFoXDTA3MDEzMTIzNTk1OVow
                                  MDELMAkGA1UEBhMCR0IxDDAKBgNVBAoTA0lCTTETMBEGA1UEAxMKV2lsbCBZYXRlczCBnzANBgkq
                                  hkiG9w0BAQEFAAOBjQAwgYkCgYEArsRj/n+3RN75+jaxuOMBWSHvZCB0egv8qu2UwLWEeiogePsR
                                  6Ku4SuHbBwJtWNr0xBTAAS9lEa70yhVdppxOnJBOCiERg7S0HUdP7a8JXPFzA+BqV63JqRgJyxN6
                                  msfTAvEMR07LIXmZAte62nwcFrvCKNPCFIJ5mkaJ9v1p7jkCAwEAAaOBrTCBqjA/BglghkgBhvhC
                                  AQ0EMhMwR2VuZXJhdGVkIGJ5IHRoZSBTZWN1cml0eSBTZXJ2ZXIgZm9yIHovT1MgKFJBQ0YpMDgG
                                  ZQVRFU0BVSy5JQk0uQ09ggdJQk0uQ09NhgtXV1cuSUJNLkNPTYcECRRlBjAO
 </wsse:BinarySecurityToken>
 <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
  <ds:SignedInfo xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
                 xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
                 xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
                 xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
   <ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#">
    <c14n:InclusiveNamespaces xmlns:c14n="http://www.w3.org/2001/10/xml-exc-c14n#" PrefixList="ds wsu xenc SOAP-ENV "/>
   </ds:CanonicalizationMethod>
   <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
   <ds:Reference URI="#TheBody">
    <ds:Transforms>
     <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#">
       <c14n:InclusiveNamespaces xmlns:c14n="http://www.w3.org/2001/10/xml-exc-c14n#" PrefixList="wsu SOAP-ENV "/>
     </ds:Transform>
    </ds:Transforms>
    <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/> 2
    <ds:DigestValue>QORZEA+gpafluShspHxhrjaFlXE=</ds:DigestValue> 3
   </ds:Reference>
  </ds:SignedInfo>
  <ds:SignatureValue>drDH0XESiyN6YJm27mfK1ZMG4Q4IsZqQ9N9V6kEnw2lk7aM3if77XNFnyKS4deglbC3ga11kkaFJ 4
                     p4jLOmYRqqycDPpqPm+UEu7mzfHRQGe7H0EnFqZpikNqZK5FF6fvYlv2JgTDPwrOSYXmhzwegUDT
                     lTVjOvuUgXYrFyaO3pw=</ds:SignatureValue>
   <ds:KeyInfo>
    <wsse:SecurityTokenReference>
      <wsse:Reference URI="#x509cert00"
                      ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509"/> 5
    </wsse:SecurityTokenReference>
   </ds:KeyInfo>
  </ds:Signature>
 </wsse:Security>
</SOAP-ENV:Header>
<SOAP-ENV:Body xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd" wsu:Id="TheBody">
 <getVersion xmlns="http://msgsec.wssecfvt.ws.ibm.com"/>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

1. バイナリー・セキュリティー・トークンには、base64binary エンコードの X.509 証明書が含まれています。 このエンコードには、SOAP メッセージの意図した受信側が署名を検証するために使用する公開鍵が格納されています。
2. メッセージ・ダイジェストを生成するためにハッシュ・プロセス中に使用されるアルゴリズム。
3. メッセージ・ダイジェストの値。
4. その後、ダイジェスト値はユーザーの秘密鍵で暗号化され、署名値としてここに含められます。
5. 署名を検証するために使用される公開鍵を含むバイナリー・セキュリティー・トークンを参照します。

暗号化された SOAP メッセージの CICS サポート

インバウンド・メッセージでは、CICS は、SOAP 本体内の暗号化済みエレメント、および本体も暗号化された暗号化済みの SOAP ヘッダー・ブロックを暗号化解除することができます。アウトバウンド・メッセージでは、CICS は SOAP 本体全体を暗号化します。

SOAP メッセージは、<Envelope> エレメントから構成される XML 文書です。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。

WSS: SOAP Message Security 仕様では、<Header> エレメントの内容の一部と <Body> エレメントのすべての内容をエレメント・レベルで暗号化することができます。つまり、あるメッセージでは、個々のエレメントを異なるレベルで暗号化したり、別のアルゴリズムを使用して暗号化したりすることができます。 例えば、オンライン購入アプリケーションで使用される SOAP メッセージでは、個々のクレジット・カードの詳細が機密のままになるように、詳細を暗号化するのが適しています。 しかし、メッセージ全体の暗号化にかかるオーバーヘッドを避けるために、一部の情報は安全性の低い (しかし高速な) アルゴリズムを使用して暗号化し、一部の情報は暗号化しないでおくことも可能であるかもしれません。

インバウンド・メッセージでは、CICS 提供のセキュリティー・メッセージ・ハンドラーが、SOAP <Body> 内の個々のエレメントを暗号化解除して、SOAP 本体も暗号化されている場合は SOAP <Header> 内のエレメントを暗号化解除することができます。 セキュリティー・メッセージ・ハンドラーは、以下のエレメントを常に暗号化解除します。

- <Header> エレメント内に検出される各エレメント (見つかった順序で)。
- SOAP <Body> エレメント内のエレメント。 暗号化された <Body> を含まない SOAP メッセージを拒否する場合は、<expect_encrypted_body> エレメントを使用して暗号化された本体を要求するようハンドラーを構成します。

アウトバウンド・メッセージでは、セキュリティー・メッセージ・ハンドラーがサポートするのは、SOAP <Body> の内容の暗号化だけです。<Header> エレメント内のエレメントは暗号化されません。 セキュリティー・メッセージ・ハンドラーが <Body> エレメントを暗号化すると、本体内のすべてのエレメントが、同じアルゴリズムと同じ鍵を使用して暗号化されます。アルゴリズム、および鍵に関する情報は、ハンドラーの構成情報で指定されます。

暗号化アルゴリズム

CICS は、XML 暗号化仕様で必要な暗号化アルゴリズムをサポートします。 それぞれのアルゴリズムは、汎用リソース ID (URI) で識別されます。

アルゴリズム	**URI**
Triple Data Encryption Standard algorithm (Triple DES)	http://www.w3.org/2001/04/xmlenc#tripledes-cbc
Advanced Encryption Standard (AES) アルゴリズム (鍵の長さは 128 ビット)	http://www.w3.org/2001/04/xmlenc#aes128-cbc
Advanced Encryption Standard (AES) アルゴリズム (鍵の長さは 192 ビット)	http://www.w3.org/2001/04/xmlenc#aes192-cbc
Advanced Encryption Standard (AES) アルゴリズム (鍵の長さは 256 ビット)	http://www.w3.org/2001/04/xmlenc#aes256-cbc

暗号化された SOAP メッセージの例

これは、CICS によって暗号化された SOAP メッセージの例です。

```
<?xml version="1.0" encoding="UTF8"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Header>
 <wsse:Security xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
                xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd"
                xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
                xmlns:xenc="http://www.w3.org/2001/04/xmlenc#" SOAP-ENV:mustUnderstand="1">

  <wsse:BinarySecurityToken
                EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#Base64Binary" 1
                ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509"
                wsu:Id="x509cert00">MIIChDCCAe2gAwIBAgIBADANBgkqhkiG9w0BAQUFADAwMQswCQYDVQQGEwJHQjEMMAoGA1UEChMD
                                   SUJNMRMwEQYDVQQDEwpXaWxsIFlhdGVzMB4XDTA2MDEzMTAwMDAwMFoXDTA3MDEzMTIzNTk1OVow
                                   MDELMAkGA1UEBhMCR0IxDDAKBgNVBAoTA0lCTTETMBEGA1UEAxMKV2lsbCBZYXRlczCBnzANBgkq
                                   hkiG9w0BAQEFAAOBjQAwgYkCgYEArsRj/n+3RN75+jaxuOMBWSHvZCB0egv8qu2UwLWEeiogePsR
                                   6Ku4SuHbBwJtWNr0xBTAAS9lEa70yhVdppxOnJBOCiERg7S0HUdP7a8JXPFzA+BqV63JqRgJyxN6
                                   msfTAvEMR07LIXmZAte62nwcFrvCKNPCFIJ5mkaJ9v1p7jkCAwEAAaOBrTCBqjA/BglghkgBhvhC
                                   AQ0EMhMwR2VuZXJhdGVkIGJ5IHRoZSBTZWN1cml0eSBTZXJ2ZXIgZm9yIHovT1MgKFJBQ0YpMDgG
                                   A1UdEQQxMC+BEVdZQVRFU0BVSy5JQk0uQ09NggdJQk0uQ09NhgtXV1cuSUJNLkNPTYcECRRlBjAO
                                   BgNVHQ8BAf8EBAMCAfYwHQYDVR0OBBYEFMiPX6VZKP5+mSOY1TLNQGVvJzu+MA0GCSqGSIb3DQEB
                                   BQUAA4GBAHdrS409Jhoe67pHL2gs7x4SpV/NOuJnn/w25sjjop3RLgJ2bKtK6RiEevhCDim6tnYW
                                   NyjBL1VdN7u5M6kTfd+HutR/HnIrQ3qPkXZK4ipgC0RWDJ+8APLySCxtFL+J0LN9Eo6yjiHL68mq
                                   uZbTH2LvzFMy4PqEbmVKbmA87alF
  </wsse:BinarySecurityToken>
  <xenc:EncryptedKey xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
   <xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5"/> 2
   <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
    <wsse:SecurityTokenReference>
     <wsse:Reference URI="#x509cert00"
                    ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509"/> 3
    </wsse:SecurityTokenReference>
   </ds:KeyInfo>
   <xenc:CipherData>
    <xenc:CipherValue>M6bDQtJrvX0pEjAEIcf6bq6MP3ySmB4TQOa/B5UlQj1vWjD56V+GRJbF7ZCES5ojwCJHRVKW1ZB5 4
                      Mb+aUzSWlsoHzHQixc1JchgwCiyIn+E2TbG3R9m0zHD3XQsKTyVaOTlR7VPoMBd1ZLNDIomxjZn2
                      p7JfxywXkObcSLhdZnc=</xenc:CipherValue>
   </xenc:CipherData>
   <xenc:ReferenceList>
    <xenc:DataReference URI="#Enc1"/>
   </xenc:ReferenceList>
  </xenc:EncryptedKey>
 </wsse:Security>
</SOAP-ENV:Header>
<SOAP-ENV:Body>
 <xenc:EncryptedData xmlns:xenc="http://www.w3.org/2001/04/xmlenc#" Id="Enc1" Type="http://www.w3.org/2001/04/xmlenc#Content">
  <xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#tripledes-cbc"/> 5
  <xenc:CipherData>
   <xenc:CipherValue>kgvqKnMcgIUn7rl1vkFXF0g4SodEd3dxAJo/mVN6ef211B1MZelg7OyjEHf4ZXwlCdtOFebIdlnK 6
                     rrksql1Mpw6So7ID8zav+KPQUKGm4+E=</xenc:CipherValue>
  </xenc:CipherData>
 </xenc:EncryptedData>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

1. バイナリー・セキュリティー・トークンには、base64binary エンコードの X.509 証明書が含まれています。 このエンコードには、対称鍵の暗号化に使用された公開鍵が格納されています。
2. 対称鍵の暗号化に使用されたアルゴリズムを提示します。
3. 対称鍵の暗号化に使用された公開鍵を含むバイナリー・セキュリティー・トークンを参照します。
4. メッセージの暗号化に使用された暗号化済みの対称鍵。
5. メッセージの暗号化に使用された暗号化アルゴリズム。
6. 暗号化されたメッセージ。

Web Services Security に合わせた RACF の構成

アウトバウンド SOAP メッセージに署名して暗号化するための公開鍵と秘密鍵のペアおよび X.509 証明書を作成して、署名および暗号化されたインバウンド SOAP メッセージを認証して暗号化解除するには、RACF などの外部セキュリティー・マネージャーを構成する必要があります。

始める前に

この作業を実行するには、その前に、CICS で作業するよう RACF をセットアップしておく必要があります。 Web サービス・パイプラインを含む CICS 領域の **DFLTUSER**、**KEYRING**、および **SEC=YES** の各システム初期化パラメーターを指定します。

注**:** 同じ **KEYRING** 上の同じ識別名を持つ複数の証明書はサポートされていません。

手順

1. 署名されたインバウンド SOAP メッセージを認証するには、次のようにします。
   a. X.509 証明書を ICSF 鍵として RACF にインポートします。
   b. **RACDCERT** コマンドを使用して、**KEYRING** システム初期化パラメーターで指定した鍵リングに証明書を添付します。

      ```
      RACDCERT ID(userid1)
      CONNECT(ID(userid2) LABEL('label-name') RING(ring-name)
      ```

      ここで、

      - *userid1* は、鍵リングのデフォルトのユーザー ID であるか、他のユーザー ID の鍵リングに証明書を添付する権限を持っています。
      - *userid2* は、証明書に関連付けるユーザー ID です。
      - *label-name* は、証明書の名前です。
      - *ring-name* は、**KEYRING** システム初期化パラメーターで指定された鍵リングの名前です。

   c. オプション: 宣言 ID を使用する場合は、証明書に関連付けられたユーザー ID が、作業を他のユーザー ID の下で実行できる代理権限を持っていることを確認します。 また、SOAP メッセージ・ヘッダーに含まれる追加の証明書も忘れずに RACF にインポートします。

   SOAP メッセージのヘッダーには、証明書または証明書への参照のいずれかが入ったバイナリー・セキュリティー・トークンを含めることができます。 この参照は、KEYNAME (RACF での証明書ラベル)、ISSUER と SERIAL 番号の組み合わせ、または SubjectKeyIdentifier です。 SubjectKeyIdentifier が RACF における証明書の定義で属性として指定された場合、CICS が認識できるのは SubjectKeyIdentifier だけです。
2. アウトバウンド SOAP メッセージに署名するには、次のようにします。
   a. 次の **RACDCERT** コマンドを使用して、X.509 証明書および公開鍵と秘密鍵のペアを作成します。

      ```
      RACDCERT ID(userid2) GENCERT
      SUBJECTSDN(CN('common-name')
                 T('title')
                 OU('organizational-unit')
                 O('organization')
                 L('locality')
                 SP('state-or-province')
                 C('country'))
      WITHLABEL('label-name')
      ```

ここで、*userid2* は、証明書に関連付けるユーザー ID です。 証明書の *label-name* 値を指定する場合は、次の文字は使用しないでください。

```
< > : ! =
```

b. **KEYRING** システム初期化パラメーターで指定した鍵リングに証明書を添付します。 **RACDCERT** コマンドを使用します。

c. 証明書をエクスポートして、SOAP メッセージの意図した受信側に発行します。

CICS が、署名を検証するために、意図した受信側の SOAP メッセージ・ヘッダーのバイナリー・セキュリティー・トークンに X.509 証明書を自動的に含めるように、パイプライン構成ファイルを編集することができます。

3. 暗号化されたインバウンド SOAP メッセージを暗号化解除するには、SOAP メッセージに、鍵ペアの一部である公開鍵が含まれている必要があります。この場合、秘密鍵は CICS で定義されます。

a. 暗号化のために、RACF で公開鍵と秘密鍵のペアおよび証明書を生成します。 鍵ペアと証明書は、ICSF を使用して生成する必要があります。

b. **KEYRING** システム初期化パラメーターで指定した鍵リングに証明書を添付します。 **RACDCERT** コマンドを使用します。

c. 証明書をエクスポートして、暗号化解除する SOAP メッセージの生成プログラムに発行します。

その後、SOAP メッセージの生成プログラムは、公開鍵を含む証明書をインポートして、これを使用して SOAP メッセージを暗号化することができます。 SOAP メッセージのヘッダーには、公開鍵またはこの公開鍵への参照のいずれかが入ったバイナリー・セキュリティー・トークンを含めることができます。 この参照は、KEYNAME、ISSUER と SERIAL 番号の組み合わせ、または SubjectKeyIdentifier です。 SubjectKeyIdentifier が RACF における公開鍵の定義で属性として指定された場合、CICS が認識できるのは SubjectKeyIdentifier だけです。

4. アウトバウンド SOAP メッセージを暗号化するには、次のようにします。

a. 暗号化に使用する公開鍵を含む証明書を ICSF 鍵として RACF にインポートします。 意図した受信側が SOAP メッセージを暗号化解除するには、公開鍵に関連付けられた秘密鍵が必要です。

b. **KEYRING** システム初期化パラメーターで指定した鍵リングに、公開鍵を含む証明書を添付します。 **RACDCERT** コマンドを使用します。

CICS は、証明書の公開鍵を使用して、SOAP 本体を暗号化し、SOAP メッセージ・ヘッダー内にバイナリー・セキュリティー・トークンとして公開鍵を含む証明書を送信します。 公開鍵は、パイプライン構成ファイルで定義されます。

次のタスク

アウトバウンド・メッセージに署名して暗号化するこの構成では、使用される証明書が CICS 領域のユーザー ID によって所有されている必要があります。 RACF では、証明書の所有者だけが秘密鍵 (署名または暗号化のプロセスで使用される) を抽出できるため、証明書は CICS 領域ユーザー ID によって所有される必要があります。

CICS が所有していない証明書を使用してメッセージの署名または暗号化を行う必要がある場合、Using an existing certificate that is not owned by the CICS region user IDの説明に従って、単一の証明書を複数の CICS システム間で共用できます。

ID 伝搬のためのプロバイダー・モードの Web サービスの構成

Web サービス要求での ID 伝搬は、トラスト・ベースの構成 (WebSphere DataPower からのクライアント認証 SSL 接続の使用など) に依存します。このタスクでは、トラステッド・クライアントから送信される ICRX ID トークンを WS-Security ヘッダーで受け取るように、PIPELINE リソースを構成します。

始める前に

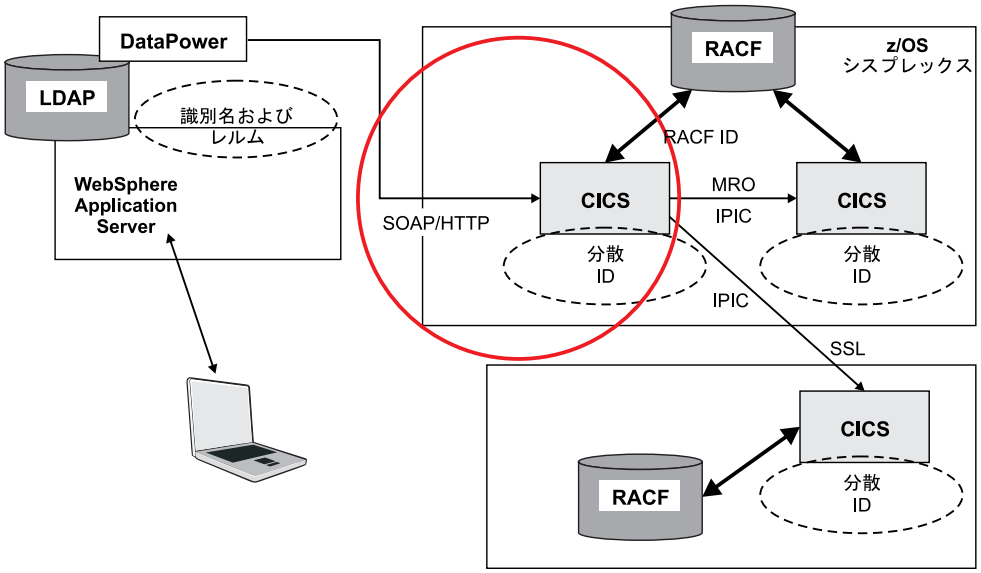
Web サービス接続を構成する前に、RACF RACMAP 設定を構成する必要があります。そうしないと、RACF へ送信されるマップされていない要求ごとに、RACF ICH408I メッセージを受け取ることになります。 RACF **RACMAP** コマンドの構成について詳しくは、ID 伝搬のための RACF の構成を参照してください。

WebSphere DataPower アプライアンスと CICS の間で信頼 関係を構成する必要があります (例えば、WebSphere DataPower と CICS の間で SSL クライアント認証を使用するなど)。 WebSphere DataPower の認証のために使用するデジタル証明書をユーザー ID に関連付け、そのユーザー ID に宣言 ID への代理権限を付与する必要があります。 代理権限について詳しくは、代理ユーザー・セキュリティーを参照してください。

このタスクについて

このタスクでは、CICS と WebSphere DataPower アプライアンスを使用して、安全かつ堅固な方法で分散 ID を伝搬できる Web サービス構成を提供する方法について説明します。 図の中の円は、このタスクが CICS 固有の構成について説明していることを示しています。

図 *29. WebSphere DataPower* からの *ICRX ID* トークンを受け入れるように *CICS* を構成する作業



WebSphere DataPower は CICS と他のアプリケーションの間の中間ノードの役割を果たします。 リモート Web サービス・リクエスター・アプリケーションは、SOAP プロトコルを使用して WebSphere DataPower アプライアンスに接続します。 WebSphere DataPower は、リモート・クライアントによって提供される資格情報を認証し、ユーザーの分散 ID を識別する z/OS ICRX ID トークンへの資格情報のマッピングを行います。 その後 SOAP メッセージは、WS-Security ヘッダーの ICRX ID トークンを使用し、トラステッド SSL 接続を介して CICS に転送されます。 ICRX ID トークンについて詳しくは、z/OS Security Server RACF Data Areas を参照してください。

CICS は、WebSphere DataPower から SOAP メッセージを受け取ります。 PIPELINE 構成ファイルは、ブラインド・トラストを指定します。これは、考えられるクライアントが WebSphere DataPower アプライアンスだけであり、WebSphere DataPower はセキュア SSL 接続を介して CICS と通信するためです。 そのため、PIPELINE 構成ファイルで追加の認証を指定する必要はありません。 WS-Security ハンドラー・プログラムは、WS-Security ヘッダーにある最初の ICRX を探し、その ICRX を使用して、ユーザーを識別します。

手順

1. PIPELINE リソースを作成するか、または既存の PIPELINE リソースを編集して、basic-ICRX モードを指定します。これを使用することにより、PIPELINE で ICRX を受け取ることができます。 ブラインド・トラストと basic-ICRX モードは、最も一般的な組み合わせです。 PIPELINE リソース・エレメントについて詳しくは、<authentication> エレメントを参照してください。

   以下は、ブラインド・トラストと basic-ICRX モードを示す PIPELINE 構成ファイルの例です。

```
<?xml version="1.0" encoding="EBCDIC-CP-US"?>
<provider_pipeline xmlns="http://www.ibm.com/software/htp/cics/pipeline">
   <service>
```

```
    <service_handler_list>
      <wsse_handler>
        <dfhwsse_configuration version="1">
          <authentication trust="blind" mode="basic-ICRX"/>
        </dfhwsse_configuration>
      </wsse_handler>
    </service_handler_list>
    <terminal_handler>
      <cics_soap_1.2_handler/>
    </terminal_handler>
  </service>
  <apphandler>DFHPITP</apphandler>
</provider_pipeline>
```

以下は、ブラインド・トラストを使用した、ICRX ID を持つ SOAP メッセージの例です。

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Header>
    <wsse:Security
        xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd"
        SOAP-ENV:mustUnderstand="1">

      <wsse:BinarySecurityToken EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss
-soap-message-security-1.0#Base64Binary"
        wsu:Id="ICRX"
        xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wsswssecurity-utility-1.0.xsd"
        ValueType="http://www.IBM.com/xmlns/prod/zos/saf#ICRXV1">

        ICRX IS HERE

      </wsse:BinarySecurityToken>

    </wsse:Security>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>

   APPLICATION SPECIFIC XML IS HERE

  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

2. WebSphere DataPower が ICRX 情報を送信できるように構成されていることを確認します。 ID 伝搬を使用するためのネットワーク・トポロジーのサンプル (Sample network topologies for using identity propagation) を参照してください。

タスクの結果

クライアント認証 SSL 接続を介して接続し、WS-Security ヘッダーの ICRX ID トークンを使用して WebSphere DataPower から出される Web サービス要求が、正常に作動するようになります。

Web Services Security に合わせたパイプラインの構成

Web Services Security (WSS) をサポートするようにパイプラインを構成するには、パイプライン構成ファイルにセキュリティー・ハンドラーを追加する必要があります。 説明に従って CICS 付属のセキュリティー・ハンドラーを使用することも、独自に作成することもできます。

始める前に

CICS 提供のセキュリティー・ハンドラーを定義する前に、WSS に関する構成情報の追加先となるパイプライン構成ファイルを指定または作成する必要があります。

手順

1. <wsse_handler> エレメントをパイプラインに追加します。 サービス・プロバイダー・パイプラインまたはサービス・リクエスター・パイプライン内の <service_handler_list> エレメントにハンドラーを含める必要があります。 次のエレメントをコーディングします。

   ```
   <wsse_handler>
     <dfhwsse_configuration version="1">

     </dfhwsse_configuration>
   </wsse_handler>
   ```

   <dfhwsse_configuration> エレメントは、構成内の他のエレメントのコンテナーです。

2. オプション: <authentication> エレメントをコーディングします。
   - サービス・リクエスター・パイプラインでは、<authentication> エレメントが、アウトバウンド SOAP メッセージのセキュリティー・ヘッダーで使用する必要がある認証のタイプを指定します。
   - サービス・プロバイダー・パイプラインでは、このエレメントが、CICS がインバウンド SOAP メッセージでセキュリティー・トークンを使用して、処理が行われるユーザー ID を決定するかどうかを指定します。

   a. **trust** 属性をコーディングして、宣言 ID を使用するかどうか、およびサービス・プロバイダーとサービス・リクエスター間の信頼関係の性質を指定します。 **trust** 属性について詳しくは、<authentication> エレメントを参照してください。

   b. オプション: **trust=none** を指定した場合は、**mode** 属性をコーディングして、メッセージで見つかった資格情報の処理方法を指定します。 **mode** 属性について詳しくは、<authentication> エレメントを参照してください。

   c. 以下のエレメントを <authentication> エレメント内にコーディングします。

      1) オプションの、空の <suppress/> エレメント。

         このエレメントがサービス・プロバイダー・パイプラインに指定される場合、ハンドラーは、作業が行われるユーザー ID を決定するメッセージ内のどのセキュリティー・トークンの使用も試みません。

         このエレメントがサービス・リクエスター・パイプラインに指定される場合、ハンドラーは、アウトバウンド SOAP メッセージに、認証に必要な、どのセキュリティー・トークンの追加も試みません。

      2) リクエスター・パイプラインでは、SOAP メッセージの本文の署名に使用されるアルゴリズムの URI を指定する、オプションの <algorithm> エレメント。trust 属性と mode 属性の値の組み合わせが、メッセージが署名されていることを示している場合は、このエレメントを指定する必要があります。 このエレメントでは、SHA1 を使用

する RSA アルゴリズムだけを指定できます。 URI は http://www.w3.org/2000/09/xmldsig#rsa-sha1 です。

3) オプションとして、<certificate_label> エレメントを組み込めます。このエレメントでは、RACF にインストールされている X.509 デジタル証明書に関連付けるラベルを指定できます。このエレメントがサービス・リクエスター・パイプラインに指定され、<suppress> エレメントが指定されない場合は、証明書が SOAP メッセージのセキュリティー・ヘッダーに追加されます。 <certificate_label> エレメントを指定しない場合は、 CICS が RACF 鍵リングでデフォルトの証明書を使用します。

このエレメントはサービス・プロバイダー・パイプラインでは無視されます。

3. オプション: <authentication> エレメントの代わりに <sts_authentication> エレメントをコーディングします。 両方をパイプライン構成ファイルにコーディングすることはできません。 このエレメントは、Security Token Service (STS) が認証に使用されることを指定し、送信される要求のタイプを決定します。
   a. オプション: サービス・プロバイダー・モードの場合のみ、**action** 属性をコーディングして、STS がセキュリティー・トークンを検証または交換するかどうかを指定します。 **action** 属性について詳しくは、<sts_authentication> エレメントを参照してください。
   b. 以下のエレメントを <sts_authentication> エレメント内にコーディングします。
      1) <auth_token_type> エレメント。 このエレメントは、サービス・リクエスター・パイプラインで <sts_authentication> エレメントを指定する場合は必須で、サービス・プロバイダー・パイプラインではオプションです。詳しくは、<auth_token_type> エレメントを参照してください。
         - サービス・リクエスター・パイプラインでは、<auth_token_type> エレメントは、CICS が DFHWS-USERID コンテナーに含まれるユーザー ID を STS に送信したときに、STS が発行するトークンのタイプを示します。 CICS が STS から受け取るトークンは、アウトバウンド・メッセージのヘッダーに置かれます。
         - サービス・プロバイダー・パイプラインでは、<auth_token_type> エレメントは、CICS がメッセージ・ヘッダーから取得して、交換または妥当性検査のために STS に送信する識別トークンを判別するために使用されます。 CICS は最初に、メッセージ・ヘッダーで指定されたタイプの識別トークンを使用します。 このエレメントを指定しない場合、CICS は、メッセージ・ヘッダーで見つけた最初の識別トークンを使用します。 CICS は、次のものは ID トークンと見なしません。
           - wsu:Timestamp
           - xenc:ReferenceList
           - xenc:EncryptedKey
           - ds:Signature

2) サービス・プロバイダー・パイプラインの場合に限り、オプションの空の <suppress/> エレメント。このエレメントが指定される場合、ハンドラーは、操作の実行に使われるユーザー ID を判別する目的でメッセージ内のどのセキュリティー・トークンの使用も試みません。<suppress/> エレメントは、STS によって戻される ID トークンが含まれます。

4. オプション: <sts_endpoint> エレメントをコーディングします。 このエレメントは、<sts_authentication> エレメントを指定した場合にのみ使用してください。 <sts_endpoint> エレメント内で、以下のエレメントをコーディングします。
   - <endpoint> エレメント。 このエレメントには、ネットワーク上の Security Token Service (STS) の場所を指し示す URI が含まれます。 STS への接続を安全に保つには、HTTP ではなく、SSL または TLS を使用することをお勧めします。

     SAML サポートを使用するには、エンドポイントを cics://PROGRAM/DFHSAML に設定します。

     また、WebSphere MQ エンドポイントは、JMS フォーマットの URI を使用して指定することもできます。
   - オプションの <jvmserver> エレメント。このエレメントは、SAML トークン・サービスを実行するよう構成された JVM サーバーを識別します。このエレメントが含まれていない場合、デフォルトのサンプル・リソース JVM サーバー DFHXSTS が想定されます。このエレメントは、SAML を使用する場合にのみ有効です。その他の状況でこれを使用した場合、エラーが発生します。
5. オプション: インバウンド SOAP メッセージにデジタル署名する必要がある場合は、空の <expect_signed_body/> エレメントをコーディングします。

   <expect_signed_body/> エレメントは、インバウンド・メッセージの <body> に署名が必要であることを示します。 インバウンド・メッセージの本文が正しく署名されていない場合、CICS はセキュリティー障害でメッセージを拒否します。
6. オプション: デジタル署名されたインバウンド SOAP メッセージを拒否する場合は、空の <reject_signature/> エレメントをコーディングします。
7. オプション: インバウンド SOAP メッセージを暗号化する必要がある場合は、空の <expect_encrypted_body/> エレメントをコーディングします。

   <expect_encrypted_body/> > エレメントは、インバウンド・メッセージの <body> を暗号化する必要があることを示します。インバウンド・メッセージの本文が正しく暗号化されていない場合、CICS はセキュリティー障害でメッセージを拒否します。
8. 部分的に、または完全に暗号化されたインバウンド SOAP メッセージを拒否する場合は、空の <reject_encryption/> エレメントをコーディングします。
9. オプション: アウトバウンド SOAP メッセージに署名する必要がある場合は、<sign_body> エレメントをコーディングします。

a. <sign_body> エレメント内にある <algorithm> エレメントをコーディングします。

b. <algorithm> エレメントの後にある <certificate_label> エレメントをコーディングします。

これは、完成した <sign_body> エレメントの例です。

```
<sign_body>
  <algorithm>http://www.w3.org/2000/09/xmldsig#rsa-sha1</algorithm>
  <certificate_label>SIGCERT01</certificate_label>
</sign_body>
```

10. オプション: アウトバウンド SOAP メッセージを暗号化する必要がある場合は、<encrypt_body> エレメントをコーディングします。

a. <encrypt_body> エレメント内にある <algorithm> エレメントをコーディングします。

b. <algorithm> エレメントの後にある <certificate_label> エレメントをコーディングします。

これは、完成した <encrypt_body> エレメントの例です。

```
<encrypt_body>
  <algorithm>http://www.w3.org/2001/04/xmlenc#tripledes-cbc</algorithm>
  <certificate_label>ENCCERT02</certificate_label>
</encrypt_body>
```

例

次の例は、ほとんどのオプション・エレメントが存在する、完成したセキュリティー・ハンドラーを示しています。

```
<wsse_handler>
    <dfhwsse_configuration version="1">
      <authentication trust="signature" mode="basic">
        <suppress/>
        <certificate_label>AUTHCERT03</certificate_label>
      </authentication>
      <expect_signed_body/>
      <expect_encrypted_body/>
      <sign_body>
        <algorithm>http://www.w3.org/2000/09/xmldsig#rsa-sha1</algorithm>
        <certificate_label>SIGCERT01</certificate_label>
      </sign_body>
      <encrypt_body>
        <algorithm>http://www.w3.org/2001/04/xmlenc#tripledes-cbc</algorithm>
        <certificate_label>ENCCERT02</certificate_label>
      </encrypt_body>
    </dfhwsse_configuration>
</wsse_handler>
```

カスタムのセキュリティー・ハンドラーの作成

独自のセキュリティー手順および処理を使用する場合は、カスタムのメッセージ・ハンドラーを作成して、セキュアな SOAP メッセージをパイプラインで処理することができます。

始める前に

ご使用のセキュリティー・ハンドラーでサポートするセキュリティーのレベルを決定し、サポートされないセキュリティーをメッセージが含んでいる場合には、必ず

該当する SOAP 障害を戻すようにする必要があります。

このタスクについて

メッセージ・ハンドラーは、インバウンドおよびアウトバウンド・メッセージでのセキュリティーも処理できる必要があります。

手順

1. **EXEC CICS GET CONTAINER** コマンドを使用して、DFHREQUEST または DFHRESPONSE コンテナーを取り出します。
2. XML を構文解析して、WS-Security メッセージ・ヘッダーにあるセキュリティー・トークンを検出します。 ヘッダーの先頭は、<wsse:Security> エレメントです。セキュリティー・トークンは、ユーザー名およびパスワード、デジタル証明書、または暗号鍵である可能性があります。 メッセージは、セキュリティー・ヘッダーに多くのトークンを持つことがあるので、ハンドラーは、処理対象であるトークンを正しく識別する必要があります。
3. メッセージに実装されているセキュリティーに応じて、適切な処理を実行します。
   a. Kerberos トークンの基本認証を実行するには **EXEC CICS VERIFY TOKEN** コマンドを発行します。 このコマンドは、提供された Kerberos トークンが有効であることを検査します。 コマンドが成功した場合は、DFHWS-USERID コンテナーを **EXEC CICS PUT CONTAINER** で更新します。その他の場合は、**EXEC CICS SOAPFAULT CREATE** コマンドを実行します。
   b. パスワードまたはパスワード・フレーズの基本認証を実行するには **EXEC CICS VERIFY PHRASE** コマンドを発行します。 このコマンドは、メッセージのセキュリティー・ヘッダーにあるユーザー名およびパスワードを検査します。 コマンドが成功した場合は、DFHWS-USERID コンテナーを **EXEC CICS PUT CONTAINER** で更新します。 その他の場合は、**EXEC CICS SOAPFAULT CREATE** コマンドを実行します。
   c. Security Token Service によってトークンの範囲を交換するか検証することにより、拡張認証を実行する場合は、Trust クライアント・インターフェースを使用します。 詳しくは、 677 ページの『メッセージ・ハンドラーからの Trust クライアントの起動』を参照してください。
   d. メッセージが署名済みの場合は、デジタル証明書の資格情報を検証します。
   e. メッセージの部分が暗号化されている場合は、セキュリティー・ヘッダーの情報を使用して、メッセージを暗号化解除します。 CICS が Web Services Security 仕様に準拠する仕組みで、これを行う方法について詳しく説明しています。

タスクの結果

CICS でセキュリティー・ハンドラー・プログラムを定義し、パイプライン構成ファイルを更新して、そのプログラムが XML に正しく配置されるようにします。サービス・リクエスターのパイプライン構成ファイルでは、パイプラインの最後で実行されるように、セキュリティー・ハンドラーを構成する必要があります。サービス・プロバイダーのパイプライン構成ファイルでは、パイプラインの最初で実行されるように、セキュリティー・ハンドラーを構成する必要があります。

次のタスク

カスタムのメッセージ・ハンドラーの作成方法に関する一般情報については、IBM Redbooks: Application Development for IBM CICS Web Services を参照してください。

メッセージ・ハンドラーからの Trust クライアントの起動

CICS は、独自のメッセージ・ハンドラーを作成して Security Token Service (STS) を起動できるようにするインターフェースを備えています。 このインターフェースを使用すると、CICS 提供のセキュリティー・ハンドラーよりも高度な処理を実行することができます。

このタスクについて

セキュリティー・ハンドラーの代わりに、またはそれに追加して Trust クライアントを使用できます。 Trust クライアント・インターフェースを使用するには、次のようにします。

手順

1. 正しいトークンを、インバウンドまたはアウトバウンド・メッセージのセキュリティー・メッセージ・ヘッダーから取り出します。
2. チャネル DFHWSTC-V1 および以下の必要なコンテナーを渡す、プログラム DFHPIRT にリンクします。
   - DFHWS-STSURI。ネットワーク上の STS の位置を格納しています。
   - DFHWS-STSACTION。STS が実行する必要がある要求のタイプの URI を格納しています。 サポートされている 2 つのアクションは `issue` および `validate` です。
   - DFHWS-IDTOKEN。STS によって検証または交換される必要があるトークンを格納しています。
   - DFHWS-TOKENTYPE。STS が応答で返す必要があるトークンのタイプを格納しています。
   - DFHWS-SERVICEURI。呼び出されている Web サービス操作の URI を格納しています。

   オプションで DFHWS-XMLNS コンテナーを組み込んで、セキュリティー・トークンを格納する SOAP メッセージのネームスペースを提供することができます。 このコンテナーについては、ヘッダー処理プログラム・インターフェースで詳しく説明しています。
3. DFHPIRT は、STS からの応答によって戻ります。 成功した応答は、DFHWS-RESTOKEN コンテナーに格納されます。

   STS で要求に関する問題が発生した場合、STS は SOAP 障害を戻します。 DFHPIRT は、その SOAP 障害を DFHWS-STSFAULT コンテナーに入れます。 STS が、SOAP 障害を発行した理由を提供した場合、理由は DFHWS-STSREASON コンテナーに入れられます。

   異常終了が発生した場合、処理エラーの詳細を格納している DFHERROR コンテナーが戻されます。

メッセージ・ハンドラーは、これらの応答を処理し、エラーが発生した際には適切な処理を実行する必要があります。例えば、メッセージ・ハンドラーは、適切な SOAP 障害を Web サービス・リクエスターに戻す場合があります。

4. 必要に応じて、応答を処理します。 プロバイダー・モードでは、パイプライン処理は、メッセージがアプリケーション・ハンドラーに到達するまでに、CICS が理解できるユーザー名を DFHWS-USERID コンテナーに配置する必要があります。リクエスター・モードでは、メッセージ・ハンドラーは、正しいトークンをアウトバウンド・メッセージのセキュリティー・ヘッダーに追加する必要があります。

次のタスク

メッセージ・ハンドラーを作成した場合、CICS でそのプログラムを配置し、該当するパイプライン構成ファイルを更新します。 サービス・リクエスターのパイプラインで、パイプライン処理の最後 (ただし CICS 提供のセキュリティー・ハンドラーの前) に発生するように、メッセージ・ハンドラーを定義します。 サービス・プロバイダーのパイプラインで、パイプラインの最初 (ただし CICS 提供のセキュリティー・ハンドラーの後) に発生するように、メッセージ・ハンドラーを定義します。

z/OS Connect のセキュリティー

z/OS Connect は、WebSphere Liberty アプリケーションであり、その構成と考慮事項は、他の WebSphere Liberty アプリケーションと同じです。さらに、z/OS Connect for CICS 1.0 および z/OS Connect Enterprise Edition には、追加のセキュリティー要件があります。

z/OS Connect for CICS 1.0 および z/OS Connect Enterprise Edition は、動的サービス・ディスカバリーを可能にする RESTful 管理インターフェースを備えていますこのインターフェースは、個々の JSON サービスと同じホスト名およびポート番号でホストされます。このインターフェースを保護するための Transport Layer Security (TLS) と個々の JSON サービスを使用することが推奨されます。

デフォルトでは、z/OS Connect for CICS 1.0 および z/OS Connect Enterprise Edition へのすべてのクライアント接続は、HTTPS プロトコルを使用する必要があります。デフォルトの動作では、CICS へのクライアント認証 TLS 接続が必要です。このデフォルトをそのまま使用する場合は、クライアント証明書を SAF ユーザー ID に関連付ける必要があります。アプリケーションは、この証明書から得られた ID を使用して実行されます。

z/OS Connect for CICS 1.0 と z/OS Connect Enterprise Edition は両方とも、HTTP 基本認証プロトコルをサポートするように構成できます。このプロトコルを使用すると、クライアントは TLS と SAF ユーザー ID/パスワードとを組み合わせて使用して接続できるようになります。HTTP 基本認証をサポートできるようにするには、Liberty サーバー構成ファイル (`server.xml`) に「`<webAppSecurity allowFailOverToBasicAuth="true"/>`」という行を追加します。

z/OS Connect for CICS 1.0 と z/OS Connect Enterprise Edition のユーザーは、`zos.connect.access.roles.zosConnectAccess` EJBROLE のメンバーでなければなりません。詳細については、SAF ロール・マッピングを使用した許可

を参照してください。

Liberty についてはLiberty でのアプリケーションの許可の構成、z/OS Connect については z/OS Connect に関するセキュリティーの構成、z/OS Connect EE については z/OS Connect Enterprise Edition V3.0 製品資料内の『z/OS Connect EE に関するセキュリティーの構成』を参照してください。

詳しくは、SAF ロール・マッピングを使用した許可、およびLiberty JVM サーバーのセキュリティーの構成 (Configuring security for a Liberty JVM server)を参照してください。

z/OS Connect サービスおよび API の許可の構成

CICS セキュリティー・モデルでは、z/OS Connect for CICS 1.0 および z/OS Connect Enterprise Edition を使用してサービスおよび API の許可を構成するという点で、いくつかの追加のアクションが必要です。

このタスクについて

z/OS Connect を使用して CICS に作業を組み込む場合、処理のさまざまな段階で次の 2 つの ID が作業に関連付けられます。

- 作業の接続プロセスで、初期の一時 ID が割り振られます。
- その後、認証済みの ID が、作業の残りの部分を実行するために使用されます。

これらの ID は、設定およびシステム環境に応じて、いくつかの方法で構成できます。

手順

1. オプション: z/OS Connect の代替の初期ユーザー ID を作成します。

   デフォルトでは、初期 ID はデフォルトの CICS ユーザー ID ですが、別のユーザー ID を割り当てて、デフォルトの CICS ユーザー ID にトランザクション CPIH またはそれと同等の作業を実行する許可を与えないようにすることもできます。

   a. 代替の初期ユーザー ID に、トランザクション CPIH および z/OS Connect を介して開始されるその他のトランザクションを実行することを許可します。

      初期ユーザー ID には、サービスまたは API のターゲット・トランザクションを実行するための許可が必要です。

2. デフォルトの初期ユーザー ID を割り当てます。以下の方法のいずれかを選択することも、両方を選択することもできます。

   - z/OS Connect をホストする JVMSERVER リソースの JVM プロファイルでユーザー ID のオーバーライド値を設定します。

     以下にオーバーライドの例を示します。ここで、`ZOSCUSER` はデフォルトの初期ユーザー ID の `-Dcom.ibm.cics.jvmserver.http.userid=ZOSCUSER` です。

     注: JVM プロファイルでデフォルトの初期ユーザー ID を設定する場合は、各 URIMAP に USERID 値を指定する必要はありません。ただし、

URIMAP に USERID を指定し、かつ JVM プロファイルでオーバーライド値を指定した場合、その URIMAP に指定した USERID が優先されます。

- z/OS Connect をターゲットとする特定の URIMAP リソースに対して USERID フィールドを設定します。

  HTTP 要求が z/OS Connect で受信されると、CICS はその要求を、インストールされている URIMAP リソースと照合します。検出された URIMAP が USERID 属性を指定する場合、JVM サーバーのデフォルトの初期ユーザー ID の代わりに、そのユーザー ID が初期ユーザー ID として使用されます。

  以下に、ZOSCDEFT という名前の URIMAP リソースの構成例を示します。ここで、JVMSERVER は USAGE 値です。総称値は PATH 属性に対して設定されます。CPIH はターゲット・トランザクションです。ZOSCUSER はデフォルトの初期ユーザー ID です。

```
NAME: ZOSCDEFT
USAGE: JVMSERVER
SCHEME: HTTP
PORT: NO
HOST: *
PATH: /zosConnect/*
TRANSACTION: CPIH
USERID: ZOSCUSER
```

注: PIPELINE SCAN メカニズムを使用することによってインストールされる URIMAP リソースは、デフォルト・ユーザー ID を使用して構成されない可能性があります。このような場合、JVMSERVER でユーザー ID のオーバーライド値を指定することを考慮できます。

注: 初期ユーザー ID は WSBind ファイルに保管できます。DFHLS2JS または DFHJS2LS のユーザーは、**USERID** 入力パラメーターの値を提供できます。**USERID** パラメーターが使用されている場合、PIPELINE SCAN 中に生成されるすべての URIMAP には要求された初期ユーザー ID が含まれます。

タスクの結果

これで、CICS がサービスおよび API の URI を認識し、ターゲット・トランザクションが接続されている場合は、使用する初期ユーザー ID を関連付けるように環境が構成されました。

第 5 章 Web サービスのトラブルシューティング

CICS での Web サービスの実装時に起こる可能性がある問題は、配置プロセス中、または実行時に CICS がメッセージを変換しているときに発生することがあります。

SOAP Web サービスのトラブルシューティング

CICS での SOAP Web サービスの実装時に起こる可能性がある問題は、配置プロセス中、または実行時に CICS が SOAP メッセージを変換しているときに発生することがあります。

配置エラーの診断

配置エラーは、CICS Web サービス・アシスタントのバッチ・ジョブまたは CICS XML アシスタントのバッチ・ジョブを実行したり、CICS に PIPELINE リソースまたは WEBSERVICE リソースをインストールしたりしようとすると発生することがあります。 ここでは、最も一般的な配置エラーについて、問題の症状、原因、および解決策を含めて説明します。

このタスクについて

配置エラーが発生した場合、通常 PIPELINE リソースは DISABLED 状態でインストールされ、WEBSERVICE リソースは UNUSABLE 状態でインストールされます。

CICS Web サービス・アシスタントのバッチ・ジョブおよび CICS XML アシスタントのバッチ・ジョブに関連する情報とエラー・メッセージは、ジョブ・ログにあります。 リソースのインストールに関連するエラー・メッセージは、システム・ログにあります。

コード 0、4、8、または 12 はアシスタントによって発行され、それ以外のコードは通常、BPXBATCH、JVM、または IEBGENER によって発行されます。

BPXBATCH によって発行されるコードは 2 つの主要カテゴリーに分類されます。つまり、128 未満のコードはコマンドの失敗を示し、128 以上のコードはシグナルによって処理が終了されたことを示します。 BPXBATCH およびその戻りコードの詳細は、z/OS UNIX システム・サービス コマンド解説書を参照してください。

手順

- CICS Web サービス・アシスタントのバッチ・ジョブまたは CICS XML アシスタントのバッチ・ジョブの実行時に、戻りコード 0、4、8、または 12 を受け取ります。 戻りコードの意味は次のとおりです。
  - 0 - ジョブは正常に完了しました。
  - 4 - 警告。 ジョブは正常に完了しましたが、1 つ以上の警告メッセージが発行されました。

© Copyright IBM Corp. 2005, 2018

– 8 - 入力エラー。 ジョブが正常に完了しませんでした。 入力パラメーターの検証中に 1 つ以上のエラー・メッセージが発行されました。
– 12 - エラー。 ジョブが正常に完了しませんでした。 実行中に 1 つ以上のエラー・メッセージが発行されました。

1. ジョブ・ログで警告メッセージまたはエラー・メッセージがないか確認します。 メッセージの詳細な説明を調べてください。 通常は、問題を修正するために実行できる処置についての説明があります。
2. ジョブで各パラメーターに合った値を入力したことを確認します。 Web サービス記述のファイル名やエレメントなどのパラメーター値では、大/小文字が区別されます。
3. 正しいパラメーターの組み合わせを指定したことを確認します。 例えば、サービス・リクエスターの Web サービス・バインディング・ファイルの生成時に、DFHWS2LS に **PGMNAME** パラメーターを含めると、エラーが発生し、ジョブは正常に完了しません。

- CICS Web サービス・アシスタントのバッチ・ジョブまたは CICS XML アシスタントのバッチ・ジョブの実行時に、戻りコード 1、136、または 139 を受け取ります。 これらの戻りコードは、多くの場合使用可能なストレージが不十分なために JVM で障害が発生したことを意味します。 CICS アシスタントでは、少なくとも 300 MB の JCL 領域サイズが必要です。ただし、文書によっては 400 MB が必要です。

  1. 領域サイズを増やすか、領域サイズを `0M` に設定することを検討します。
  2. 領域サイズを制限する可能性があるアクティブな IEFUSI 出口があるかを調べます。

     注: 64 ビット JVM を使用する場合は、必ず適切な MEMLIMIT 値を指定してください。

- CICS Web サービス・アシスタントのバッチ・ジョブ DFHLS2WS または CICS XML アシスタントのバッチ・ジョブ DFHLS2SC の実行時に、戻りコード 137 を受け取ります。 この戻りコードは、ジョブがタイムアウトになったことを意味します。

  1. ジョブの EXEC ステートメントの **TIME** パラメーターを **TIME=1440** にコーディングして時間を増やすか、SYS1.PARMLIB(BPXPRMxx) メンバーの MAXCPUTIME 値を増やします。

- WEBSERVICE リソースをインストールしようとすると、DFHPI0914 エラー・メッセージを受け取ります。 このメッセージには、インストール障害の原因に関する情報が含まれています。

  1. z/OS UNIX での Web サービス・バインディング・ファイルの読み取りを CICS に許可したことを確認します。
  2. Web サービス・バインディング・ファイルが破損していないことを確認します。 これは、FTP を使用して、バイナリー・モードではなくテキスト・モードでファイルを z/OS UNIX に転送する場合に発生することがあります。
  3. 同じ名前の 2 つの Web サービス・バインディング・ファイルが異なるピックアップ・ディレクトリーにないことを確認します。
  4. Web サービス・リクエスター・アプリケーションのリソースをインストールする場合は、SOAP バインディングのバージョンがパイプラインでサポート

されるレベルと一致することを確認します。 SOAP 1.2 をサポートするサービス・リクエスター・パイプラインに SOAP 1.1 WEBSERVICE をインストールすることはできません。

5. プロバイダー・モード WEBSERVICE リソースをリクエスター・モード・パイプラインにインストールしていないことを確認します。 プロバイダー・モードの Web サービス・バインディング・ファイルでは **PROGRAM** 値が指定されるのに対して、リクエスター・モードのバインディング・ファイルではこの値は指定されません。
6. DFHWS2LS または DFHLS2WS を使用する場合は、Web サービス・バインディング・ファイルの生成時に正しいパラメーターを指定したことを確認します。 **PGMNAME** など、パラメーターの中には Web サービス・プロバイダーでしか使用できないものがあり、Web サービス・リクエスターを作成する場合には除外する必要があります。
7. DFHWS2LS または DFHLS2WS を使用する場合は、ジョブによって発行されたメッセージを調べて、WEBSERVICE リソースを作成する前に解決すべき問題があるかどうかを確認します。

- PIPELINE リソースがインストールに失敗し、DFHPI0700、DFHPI0712、または DFHPI0714 などのエラー・メッセージを受け取ります。
  1. DFHPI0700 エラー・メッセージを受け取った場合は、CICS 領域で PL/I 言語サポートを使用可能にする必要があります。 これは、PIPELINE リソースをインストールする前に行う必要があります。詳細については、PL/I の Language Environment サポートを参照してください。
  2. パイプライン構成ファイルを読み取るために z/OS UNIX ディレクトリーへのアクセスを CICS に許可したことを確認します。
  3. **WSDIR** パラメーターで指定したディレクトリーが有効なことを確認します。 z/OS UNIX ではディレクトリーとファイル名は大/小文字が区別されるため、特に大/小文字を確認します。
  4. CICS 領域に ENABLED 状態の同じ名前の PIPELINE リソースがないことを確認します。
- PIPELINE リソースが DISABLED 状態でインストールされます。 DFHPI0702 から DFHPI0711 までの範囲のエラー・メッセージを受け取ります。
  1. パイプライン構成ファイルにエラーがないことを確認します。 パイプライン構成ファイルのエレメントは、特定の場所にしか現れません。 これらのエレメントを誤って指定すると、DFHPI0702 エラー・メッセージを受け取ります。 このメッセージには、問題の原因となっているエレメントの名前が含まれています。 エレメント記述を調べて、正しい場所でコーディングしたことを確認します。
  2. タブなどの印刷不能文字がパイプライン構成ファイルにないことを確認します。
  3. XML が有効なことを確認します。 XML が無効な場合、これにより PIPELINE リソースをインストールしようとすると構文解析エラーが発生することがあります。
  4. パイプライン構成ファイルが US EBCDIC でエンコードされていることを確認します。 別の EBCDIC エンコードを使用すると、CICS がファイルを処理できません。

- WEBSERVICE リソースが DISABLED 状態です。 CICS バンドル内に定義およびインストールされた WEBSERVICE リソースでは、DISABLED 状態と DISABLING 状態だけが使用可能です。
  1. WEBSERVICE リソースに関連付けられた PIPELINE リソースが既に破棄された場合、WEBSERVICE リソースは DISABLED 状態に入ります。 PIPELINE リソースが欠落している理由を調べて、適切な場合には置き換えてください。
  2. WEBSERVICE リソースが定義されている CICS バンドルに対する使用不可アクションが既に実行された場合、Web サービスが使用されなくなった時点で WEBSERVICE リソースが DISABLED 状態に入ります。 CICS バンドルの状態を調べて、適切な場合にはバンドルを使用可能にしてください。

サービス・プロバイダーのランタイム・エラーの診断

プロバイダー・モード・パイプラインでのインバウンド・メッセージの受信または処理中に問題が発生する場合は、トランスポートまたは特定の SOAP メッセージに問題がある可能性があります。

手順

- HTTP または WebSphere MQ トランスポート・エラーが発生したことを示す、DFHPI0401 や DFHPI0502 のようなメッセージを受け取ります。 トランスポートが HTTP の場合、クライアントは「500 Server Internal Error (500 サーバーの内部エラーが発生しました)」メッセージを受け取ります。 トランスポートが WebSphere MQ の場合、メッセージは送達不能キュー (DLQ) に書き込まれます。 CICS は受け取ったメッセージのタイプを判別できないため、SOAP 障害は Web サービス・リクエスターに戻されません。
- DFH*xx* メッセージおよび「404 Not Found (404 見つかりません)」エラー・メッセージを受け取ります。
  1. Web サービス・アシスタントを使用していない場合は、URIMAP リソースを作成する必要があります。 Web サービス・アシスタントを使用している場合は、**PIPELINE SCAN** コマンドの実行時に URIMAP が自動的に作成されます。 システム・ログには、このコマンドを実行した結果発生したエラーに関する情報が記載されています。
  2. WEBSERVICE リソースを使用できること、およびこのリソースが関連付けられている URIMAP が予期した URIMAP であることを確認します。 WEBSERVICE リソースが UNUSABLE 状態にある場合は、 681 ページの『配置エラーの診断』を参照してください。
  3. URI およびポート番号を正しく指定したことを確認します。 URIMAP リソース上の属性 PATH には大/小文字の区別があるため、特に大/小文字を確認します。
- 予期せぬエラーが報告されている場合は、CEDX を使用して Web サービス・アプリケーションをデバッグすることを検討してください。
  1. システム・ログを調べて、CICS によって報告されているエラー・メッセージを確認します。 これは、発生しているエラーのタイプを示している可能性があります。 CICS がエラーを報告していない場合は、要求がネットワーク経由で CICS に到達していることを確認します。

2. HTTP トランスポートの場合は CPIH、WebSphere MQ トランスポートの場合は CPIQ、またはこれが異なる場合はユーザーが URIMAP で指定したトランザクションに対して CEDX を実行します。

   パイプライン処理中にアプリケーション・ハンドラーの前にタスク切り替えが行われると、DFHWS-TRANID コンテナーが取り込まれない限り、新規のタスクが最初のタスクと同じトランザクション ID の下で実行されます。最初のタスクの CEDX セッションにロックがあるため、これによって CEDX の実行が妨害されることがあります。 この問題は、DFHWS-TRANID を使用してタスク切り替え時にトランザクション ID を変更し、パイプラインとアプリケーションの両方のタスクで別個に CEDX を使用できるようにすることによって回避できます。CEDX について詳しくは、CEDX トランザクションの使用を参照してください。

3. CEDX が活動化されないか、問題を解決できない場合は、PI、SO、AP、EI、および XS ドメインをアクティブにして補助トレースを実行することを検討してください。 これは、CICS 領域にセキュリティーの問題、TCP/IP の問題、アプリケーション・プログラムの問題、またはパイプラインの問題があることを示している可能性があります。 例外トレース・ポイントまたは異常終了がないか調べてください。

- 変換エラーを受け取る場合は、 690 ページの『データ変換エラーの診断』を参照してください。
- 問題が MTOM メッセージに関連していると思う場合は、 688 ページの『MTOM/XOP エラーの診断』を参照してください。
- 永続 HTTP 接続が定期的に切断される場合、次のようにします。
  1. HTTP 接続のパフォーマンス・チューニングが使用可能かどうかチェックします。詳細については、SOTUNINGを参照してください。
  2. パフォーマンス・チューニングが使用可能な場合、CICS は永続 HTTP 接続を定期的に切断することで、共用 IP エンドポイントで listen している領域間で接続が再配分されるようにします。
- CICS 領域が最大容量に達して接続の試行が拒否された場合、次のようにします。
  1. HTTP 接続のパフォーマンス・チューニングが使用可能かどうかチェックします。詳細については、SOTUNINGを参照してください。
  2. パフォーマンス・チューニングが使用可能な場合、CICS が最大容量に達すると、すべてのインバウンド HTTP 接続のオープン要求は、CICS の外部の、TCP/IP 内にある TCPIPSERVICE のリスニング接続のバックログ・キューに入ります。領域の TCP/IP グローバル統計 SOG_PAUSING_HTTP_LISTENING フィールドは、それが現行のケースかどうかを反映し、 SOG_TIMES_AT_ACCEPT_LIMIT/ SOG_TIME_LAST_PAUSED_HTTP_LISTENING フィールドには、これまでの出現に関する詳細が含まれています。
  3. **NETSTAT ALL** を使用して、TCPIPSERVICE のリスニング接続の ConnectionsDropped 統計を取得します。これは、受信され、バックログ・キューで待機している接続要求の数が最大制限数に達したために除去された接続要求の数です。 詳細については、「z/OS Communications Server IP システム管理者のコマンド」の『Netstat』を参照してください。

除去された接続数に関する統計は、TCP/IP サービス: リソース統計の以下のフィールドからも確認できます。

- ドロップされた接続の数 (SOR_CONNS_DROPPED)
- 接続が最後にドロップされた時刻 (SOR_CONN_LAST_DROPPED)

4. ConnectionsDropped 値が高すぎる場合は、TCPIPSERVICE のバックログ属性の値を大きくしてください。

サービス・リクエスターのランタイム・エラーの診断

サービス・リクエスター・アプリケーションから Web サービス要求を送信する際に問題が発生するか、Web サービス・プロバイダーから SOAP 障害メッセージを受け取る場合は、このセクションを参照してください。

このタスクについて

発生する問題は、個々の Web サービスのエラーまたはトランスポート・レベルでの問題が原因になっている可能性があります。

手順

- アプリケーション・プログラムで **INVOKE SERVICE** コマンドを使用している場合、問題があると RESP コードと RESP2 コードが戻されます。
  1. 何が問題なのかを示す、INVOKE SERVICE コマンドの RESP および RESP2 コードの意味を調べます。
  2. CICS システム・ログを調べて、問題の原因を判別する際に役立つメッセージがあるかどうかを確認します。
- SOAP 要求メッセージを送信できず、パイプラインが DFHERROR コンテナーを戻す場合は、パイプラインが SOAP メッセージを処理しようとしたときに問題が発生しました。
  1. DFHERROR コンテナーの内容を調べてください。 ここには、エラー・メッセージおよび発生した問題についての説明データが含まれているはずです。
  2. パイプラインに新規のメッセージ・ハンドラーまたはヘッダー処理プログラムを導入しましたか? 導入した場合は、その新規のプログラムを除去して、Web サービスを再実行することによって、問題が解決するかどうかを確認します。 メッセージ・ハンドラーが、パイプラインに存在しないコンテナーを使用して処理を実行しようとしたり、読み取り専用のコンテナーを更新しようとしたりすると、パイプラインは処理を停止して、DFHERROR コンテナーでエラーを戻します。 ヘッダー処理プログラムが更新できるのは、パイプラインの限定されたコンテナー群のみです。詳細については、ヘッダー処理プログラム・インターフェースを参照してください。
  3. Web サービス・リクエスター・アプリケーションが Web サービス要求を送信するために使用しているコマンドが **INVOKE SERVICE** ではない場合、必要な制御コンテナーがすべて作成されていること、およびこれらのコンテナーのデータ・タイプが正しいことを確認します。特に、DFHREQUEST コンテナーのデータ・タイプが BIT ではなく CHAR であることを確認します。
  4. Web サービス・リクエスター・アプリケーションが **INVOKE SERVICE** コマンドを使用しており、RESP 値 INVREQ および RESP2 コード 14 が戻され

る場合は、データ変換エラーが発生したことを示しています。 690 ページの『データ変換エラーの診断』を参照してください。

5. パイプライン処理中に SOAP メッセージ内の XML がカスタムのメッセージ・ハンドラーによって無効にされていなかったことを確認します。 CICS は、パイプライン内のアウトバウンド・メッセージで検証を実行しません。アプリケーションが **INVOKE SERVICE** コマンドを使用する場合、SOAP メッセージの本文が DFHREQUEST コンテナー内に置かれると、XML が CICS によって生成され、適切な形式になります。ただし、SOAP メッセージの内容を変更する追加のメッセージ・ハンドラーがある場合、これはパイプラインでは検証されません。

- SOAP メッセージを送信できるのに、タイムアウト・エラーまたはトランスポート・エラーが発生する場合は、通常、これは SOAP 障害として戻されます。 プログラムが **INVOKE SERVICE** コマンドを使用している場合、CICS は、タイムアウト・エラーでは TIMEDOUT の RESP 値および RESP2 コード 2 を戻し、トランスポート・エラーでは INVREQ の RESP 値および RESP2 コード 17 を戻します。
  1. ネットワーク・エンドポイントが存在することを確認します。
  2. PIPELINE リソース上の RESPWAIT 属性が、ご使用のアプリケーションの要件を満たすよう正しく構成されていることを確認します。 RESPWAIT 属性は、CICS がアプリケーションに戻るまでに Web サービス・プロバイダーからの応答を待機する期間を定義します。 値が指定されていない場合、CICS は、HTTP ではデフォルトの 10 秒、WebSphere MQ ではデフォルトの 60 秒を使用します。ただし、CICS は、ディスパッチャーでもトランザクションごとにタイムアウトを持ちます。これが使用されているプロトコルのデフォルトより短い場合は、CICS は代わりにディスパッチャーのタイムアウトを使用します。
- SOAP メッセージを送信できるのに、Web サービス・プロバイダーから予期していなかった SOAP 障害の応答が戻される場合は、SOAP 障害の詳細について DFHWS-BODY コンテナーの内容を調べてください。
  1. DFHPIRT インターフェースを使用して、DFHREQUEST 内の完全な SOAP エンベロープを送信する場合は、アウトバウンド・メッセージに重複した SOAP ヘッダーが含まれていないか確認してください。 これは、リクエスターのパイプラインで SOAP 1.1 または SOAP 1.2 メッセージ・ハンドラーが使用された場合に発生する可能性があります。 SOAP メッセージ・ハンドラーは、サービス・リクエスター・アプリケーションによって SOAP ヘッダーが SOAP エンベロープ内にすでに指定されている場合にも、SOAP ヘッダーを追加します。 このシナリオでは、次のいずれかを行うことができます。

  – パイプラインから SOAP 1.1 または SOAP 1.2 メッセージ・ハンドラーを除去する。これは、そのパイプラインを使用している他のサービス・リクエスター・アプリケーションに影響を与えます。
  – アプリケーションによって DFHREQUEST に置かれた SOAP エンベロープから SOAP ヘッダーを除去する。 必要な SOAP ヘッダー が CICS により追加されます。 ヘッダーで追加の処理を実行する場合は、ヘッダー処理プログラム・インターフェースを使用できます。

– 代わりにアプリケーションで **WEB SEND** コマンドを使用し、Web サポートから抜ける。

- 問題が MTOM メッセージの送受信に関連していると思う場合は、『MTOM/XOP エラーの診断』を参照してください。

MTOM/XOP エラーの診断

MTOM/XOP エラーは、リクエスター・モード・パイプラインとプロバイダー・モード・パイプラインの両方で、実行時に発生する可能性があります。

始める前に

MTOM/XOP をサポートするようパイプラインを構成する際に問題が発生した場合は、681 ページの『配置エラーの診断』を参照してください。

このタスクについて

手順

- Web サービス要求メッセージを MTOM 形式で送信できるのに、Web サービス・プロバイダーから SOAP 障害メッセージを受け取る場合は、SOAP 障害の詳細について DFHWS-BODY コンテナーの内容を調べてください。
  1. Web サービス・プロバイダーは MIME Multipart/Related メッセージを受信できますか? Web サービス・プロバイダーが MTOM 形式をサポートしていない場合、戻される障害は実装によって異なることがあります。 Web サービス・プロバイダーが別の CICS アプリケーションである場合、SOAP 障害は、MIME メッセージが有効なコンテンツ・タイプではないことを示しています。
  2. Web サービス・プロバイダーが MIME メッセージを受信できる場合は、パイプラインがメッセージを直接モードで送信しているか、または互換モードで送信しているかを確認します。 MTOM メッセージを直接モードで送信すると、XML の問題が発生する可能性があります。
  3. 問題が XML に関連しているかどうかを調べるには、Web サービスの検証をオンにします。 これによって、パイプライン全体で MTOM メッセージが互換モードで処理されるようになります。 この処理の一環として、MTOM ハンドラーは base64binary データを最適化するためにメッセージの内容を解析します。 XML にエラーがある場合は、CICS はエラーを DFHERROR コンテナー内に入れて、パイプラインで MTOM トランスポート障害を発行します。
  4. DFHERROR コンテナーの内容を調べて、これが発生した問題を示しているかどうかを確認します。 この情報が問題の原因を診断するのに十分ではない場合は、レベル 2 トレースの PI ドメインを実行します。
  5. トレース・ポイント PI 0C16 を調べます。 ここには、検出した問題が詳細に説明されており、リクエスター・アプリケーションによって提供された XML の問題を修正する際に役立ちます。
- アウトバウンド MTOM メッセージに必要なバイナリー添付ファイルがない場合は、バイナリー・データが小さすぎて、バイナリー添付ファイルとして最適化できないと見なされたことを示している可能性があります。 CICS は、パイプラインでデータを最適化する処理オーバーヘッドを許容できるほど十分な大きさのデ

ータにのみバイナリー添付ファイルを作成します。 サイズが 1,500 バイトを下回るバイナリー・データは最適化されません。

- アウトバウンド MTOM メッセージを互換モードで送信できず、パイプラインが DFHERROR コンテナーを戻す場合は、パイプラインが MTOM メッセージを処理しようとしたときに問題が発生しました。
  1. DFHERROR コンテナーの内容を調べてください。 ここには、エラー・メッセージおよび発生した問題についての説明データが含まれているはずです。
  2. アウトバウンド MTOM メッセージ内の XML が有効なことを確認します。 CICS は、パイプライン内のアウトバウンド・メッセージで検証を実行しません。
- DFHPI1100E メッセージを受信する場合は、CICS が受信した MTOM メッセージの MIME ヘッダーに問題が発生しました。 CICS メッセージには、発生した MIME エラーの汎用クラスが含まれています。 発生した正確な問題を見つけるには、次のようにします。
  1. CICS 領域で補助トレースがアクティブになっている場合は、例外トレース・エントリーがないかを確認します。
  2. トレース・ポイント PI 1305 を調べます。 ここには、MIME ヘッダー・エラーの性質、ヘッダー内のエラーの場所、およびエラーの前後にある 80 バイトまでのテキストが記述されているため、エラーが発生した場所のコンテキストを理解することができます。

  例えば、次のトレースの抜粋は、MIME コンテンツ・タイプの開始パラメーターが、引用符で囲まれていないが、引用符付きストリングの外に無効な文字が含まれているため無効であることを示しています。

```
PI 1305 PIMM *EXC* - MIME_PARSE_ERROR -

            TASK-01151 KE_NUM-0214 TCB-QR   /009C7B68 RET-9C42790A TIME-10:33:41.3667303015 INTERVAL-00.0000053281       =000599=
              1-0000  C5A79785 83A38584 40978199 819485A3  859940A5 8193A485 40A39692 85954096  *Expected parameter value token o*
                0020  994098A4 96A38584 40A2A399 899587                                      *r quoted string                 *
              2-0000  D4C9D4C5 40A2A895 A381A740 85999996  994081A3 404EF0F0 F0F0F1F1 F2408995  *MIME syntax error at +0000112 in*
                0020  40C39695 A38595A3 60A3A897 85408885  81848599                            * Content-type header            *
              3-0000  5F626F75 6E646172 793B2074 7970653D  22617070 6C696361 74696F6E 2F786F70  *_boundary; type="application/xop*
                0020  2B786D6C 223B2073 74617274 2D696E66  6F3D2261 70706C69 63617469 6F6E2F73  *+xml"; start-info="application/s*
                0040  6F61702B 786D6C22 3B207374 6172743D                                      *oap+xml"; start=                *
              4-0000  3C736F61 70736C75 6E674074 6573742E  68757273 6C65792E 69626D2E 636F6D3E  *<soapslung@test.hursley.ibm.com>*
                0020  3B206368 61727365 743D7574 662D38                                      *; charset=utf-8                 *
```

- パイプライン処理はインバウンド MTOM メッセージの解析に失敗し、Web サービス・リクエスターは SOAP 障害メッセージを受け取ります。 これは、MTOM メッセージ内の XOP 文書に問題があったことを示しています。 直接モードでは、SOAP 障害はアプリケーション・ハンドラーによって生成されます。 パイプラインが互換モードで実行されている場合、メッセージは、SOAP メッセージの作成時に MTOM ハンドラーによって解析されます。 この場合、CICS は接頭部が DFHPI のエラー・メッセージと SOAP 障害を発行します。
  1. 接頭部が DFHPI のエラー・メッセージは、XOP 文書の何に問題があるかを示しています。 例えば、MIME ヘッダーが無効か、バイナリー添付ファイルがない可能性があります。
  2. 問題の正確な原因を見つけるには、例外トレース・ポイントがないか調べます。 特に、先頭文字が PI 13*xx* のトレース・ポイントを調べます。 ここには、発生した例外の詳細が記述されています。

  また、PI レベル 2 トレースを実行して、エラーをもたらした一連のイベントを設定することもできますが、これによってパフォーマンスが大きな影響を受ける可能性があるため、実動領域ではお勧めできません。

データ変換エラーの診断

データ変換エラーは、SOAP メッセージを CICS COMMAREA またはコンテナーに変換したり、COMMAREA またはコンテナーから SOAP メッセージに変換すると、実行時に発生することがあります。

始める前に

SOAP 障害メッセージおよび障害が発生したことを示す CICS メッセージが生成されるなどの症状があります。

このタスクについて

データ変換の問題が発生した場合は、以下のステップを実行します。

手順

1. WEBSERVICE リソースが最新の状態になっていることを確認します。 Web サービスの Web サービス・バインディング・ファイルを再生成して、CICS に再配置します。
2. リモート Web サービスが、CICS によって使用または生成されたものと同じバージョンの Web サービス文書 (WSDL) を使用して生成されたことを確認します。
3. WEBSERVICE リソースが現行の Web サービス・バインディング・ファイルを使用していることが確かな場合は、次のようにします。
   a. コマンド `SET WEBSERVICE(name) VALIDATION` を使用して、WEBSERVICE リソースのランタイム検証を使用可能にします。ここで、*name* は WEBSERVICE リソース名です。
   b. メッセージ・ログに CICS メッセージ DFHPI1001 または DFHPI1002 がないかどうかを確認します。 DFHPI1001 には、データ変換の問題に関する正確な性質が記載されており、変換エラーの原因を特定するのに役立ちます。 DFHPI1002 は、問題が見つからなかったことを示しています。
   c. 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。
4. それでも変換エラーの理由を特定できない場合は、障害が発生している Web サービスの CICS トレースを使用します。 次の PI ドメインの例外トレース・エントリーを探します。

   ```
   PI 0F39 - PICC   *EXC* - CONVERSION_ERROR
   PI 0F08 - PIII   *EXC* - CONVERSION_ERROR
   ```

   PICC 変換エラーは、インバウンド SOAP メッセージを COMMAREA またはコンテナーに変換する際に問題が発生したことを示しています。 PIII 変換エラーは、アプリケーション・プログラムが提供した COMMAREA またはコンテナーから SOAP メッセージを生成する際に問題が発生したことを示しています。 どちらの場合も、トレース・ポイントは、変換エラーに関連したフィールドの名前を示しており、また問題の原因となる値も示していることがあります。これらのトレース・ポイントのいずれかが発生した場合、その後に変換エラーが発生します。 これらの変換エラーの考えられる解釈については、メッセージ DFHPI1007 から DFHPI1010 の説明を参照してください。

データ変換エラーが起こる理由

CICS による SOAP メッセージの検証は、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。 これは、WSDL を使用して SOAP メッセージを正常に検証できるが、ランタイム環境では失敗することを意味します。また、その逆も同様です。

WEBSERVICE リソースは、マッピング指示をカプセル化して、 CICS が実行時にデータ変換を実行できるようにします。 WEBSERVICE リソースに記述されるように、入力と期待されるデータが一致しないと、変換エラーが起こります。

この不一致は、以下のどの理由でも起こります。

- CICS が受け取る SOAP メッセージが、WEBSERVICE リソースに関連付けられた Web サービス記述 (WSDL) について検査される際に、このメッセージが適切な形式でなく、有効でない。
- CICS が受け取る SOAP メッセージが適切な形式であり有効だが、WEBSERVICE リソースの範囲外の値が含まれている。
- COMMAREA またはコンテナーの内容と、WEBSERVICE リソースおよび Web サービスが生成された言語構造に整合性がない。

例えば、WSDL 文書で、10 から 20 の間の値しか持てない unsignedInt のような範囲制限をフィールドに指定する場合があります。 SOAP メッセージに値 25 が含まれている場合に、SOAP メッセージを検証すると、このメッセージは無効として拒否されます。 値 25 は整数については有効な値として受け入れられるため、アプリケーションに渡されます。

2 番目の例は、WSDL 文書が最大長を指定しないでストリングを指定する場合です。 DFHWS2LS は、Web サービス・バインディング・ファイルを生成する際に、デフォルトで最大長を 255 文字と仮定します。 SOAP メッセージに 300 文字が含まれている場合、最大長が設定されていないので WSDL に対する検査ではメッセージは有効とされますが、CICS によって割り振られる 255 文字のバッファーに値が合わないので、メッセージを変換しようとするとエラーが報告されます。

コード・ページの問題

CICS は、**LOCALCCSID** システム初期設定パラメーターの値を使用して、アプリケーション・プログラム・データをエンコードします。 しかし、Web サービス・バインディング・ファイルは US EBCDIC (Cp037) でエンコードされます。 このため、アプリケーション・プログラムで使用するコード・ページが、US EBCDIC コード・ページと異なる文字をエンコードすると、データ変換の問題が発生することがあります。 この問題を避けるために、Web サービス・アシスタントのバッチ・ジョブで **CCSID** パラメーターを使用して、アプリケーション・プログラムと Web サービス・バインディング・ファイルの間でデータをエンコードするのに異なるコード・ページを指定できます。このパラメーターの値は、特定の WEBSERVICE リソースについて、**LOCALCCSID** システム初期設定パラメーターを指定変更します。 **CCSID** の *value* は EBCDIC CCSID になります。

変換エラーについての SOAP 障害メッセージ

実行時に変換エラーが起こり、CICS が Web サービス・プロバイダーのように動作する場合は、サービス・リクエスターに SOAP 障害メッセージが発行されます。この SOAP 障害メッセージには、CICS が発行したメッセージが含まれます。

サービス・リクエスターは、以下の SOAP 障害メッセージのいずれかを受け取ることがあります。

- `Cannot convert SOAP message (SOAP メッセージを変換できません)`

  この障害メッセージは、SOAP メッセージが適切な形式でなく有効でないか、値が範囲外であることを、暗黙に示しています。
- `Outbound data cannot be converted (アウトバウンド・データを変換できません)`

  この障害メッセージは、COMMAREA またはコンテナーの内容に一貫性がないことを暗黙に示しています。
- `Operation not part of web service (Web サービスの操作ではありません)`

  この障害メッセージは、CICS が無効な SOAP メッセージを受け取ったときに出される、特殊な形です。

CICS が Web サービス・リクエスターである場合は、**INVOKE SERVICE** コマンドが、INVREQ の RESP コードと 14 の RESP2 値を戻します。

JSON Web サービスのトラブルシューティング

CICS での JSON Web サービスの実装時に起こる可能性がある問題は、配置プロセス中、または実行時に CICS がメッセージを変換しているときに発生することがあります。

このタスクについて

この問題判別情報は、CICS 固有のものです。IBM サポートに問題を報告する必要がある場合は、 MustGather データの情報を参照して、必要なすべての診断情報を収集してあるか確認してください。

JSON 要求のサービス・プロバイダーとしての CICS のサポートは、SOAP Web サービスの CICS サポートに厳密に基づいています。

JSON デプロイメントの問題のトラブルシューティング

デプロイメント・エラーは、PIPELINE または WEBSERVICE のいずれかのリソースを CICS にインストールしようとする場合に発生することがあります。ここでは、最もよく発生するデプロイメント・エラーについて、問題の症状、原因、解決策などを説明します。

手順

PIPELINE または WEBSERVICE リソースを CICS にインストールするときに、以下のエラーが発生する可能性があります。

- WEBSERVICE リソースをインストールしようとすると、DFHPI0914 エラー・メッセージを受け取ります。 このメッセージには、インストール障害の原因に関する情報が含まれています。
  1. z/OS UNIX での Web サービス・バインディング・ファイルの読み取りを CICS に許可したことを確認します。
  2. Web サービス・バインディング・ファイルが破損していないことを確認します。 この破損は、FTP を使用して、バイナリー・モードではなくテキスト・モードでファイルを z/OS UNIX に転送する場合に発生することがあります。
  3. 同じ名前の 2 つの Web サービス・バインディング・ファイルが異なるピックアップ・ディレクトリーにないことを確認します。
  4. プロバイダー・モード WEBSERVICE リソースをリクエスター・モード・パイプラインにインストールしていないことを確認します。 プロバイダー・モードの Web サービス・バインディング・ファイルでは **PROGRAM** 値が指定されるのに対して、リクエスター・モードのバインディング・ファイルではこの値は指定されません。
  5. DFHJS2LS または DFHLS2JS を使用している場合、ジョブによって発行されたメッセージを調べて、WEBSERVICE リソースを作成する前に解決しなければならない問題があるかどうかを確認します。
- PIPELINE リソースがインストールに失敗し、DFHPI0700、DFHPI0712、DFHPI0714、または PIPELINE リソースで障害が発生したことを示す別のエラー・メッセージを受け取ります。
  1. DFHPI0700 エラー・メッセージを受け取った場合は、CICS 領域で PL/I 言語サポートを使用可能にする必要があります。 このサポートは、PIPELINE リソースをインストールする前に行う必要があります。詳細については、PL/I の Language Environment サポートを参照してください。
  2. パイプライン構成ファイルを読み取るために z/OS UNIX ディレクトリーへのアクセスを CICS に許可したことを確認します。
  3. **WSDIR** パラメーターで指定したディレクトリーが有効なことを確認します。 z/OS UNIX ではディレクトリーとファイル名は大/小文字が区別されるため、特に大/小文字を確認します。
  4. CICS 領域に ENABLED 状態の同じ名前の PIPELINE リソースがないことを確認します。
- PIPELINE リソースが DISABLED 状態でインストールされます。 DFHPI0702 から DFHPI0711 までの範囲のエラー・メッセージを受け取ります。
  1. パイプライン構成ファイルにエラーがないことを確認します。 パイプライン構成ファイルのエレメントは、特定の場所にしか現れません。 これらのエレメントを誤って指定すると、DFHPI0702 エラー・メッセージを受け取ります。 このメッセージには、問題の原因となっているエレメントの名前が含まれています。 エレメント記述を調べて、正しい場所でコーディングしたことを確認します。
  2. 水平タブ文字などの印刷不能文字がパイプライン構成ファイルにないことを確認します。

3. XML が有効なことを確認します。 XML が無効な場合、これにより PIPELINE リソースをインストールしようとすると構文解析エラーが発生することがあります。
4. パイプライン構成ファイルが US EBCDIC でエンコードされていることを確認します。 別の EBCDIC エンコードを使用すると、CICS がファイルを処理できません。

JSON アシスタントのトラブルシューティング

JSON アシスタントで問題が発生している場合は、トラブルシューティングの手法を使用して問題を診断してください。

手順

JSON アシスタントを実行するとき、以下のエラーが発生する可能性があります。

- CICS JSON アシスタントのバッチ・ジョブを実行すると、戻りコード 0、4、8、12 のいずれかを受け取ります。 戻りコードの詳細については、 697 ページの『JSON アシスタントの戻りコード』 を参照してください。
  1. ジョブ・ログで警告メッセージまたはエラー・メッセージがないか確認します。 メッセージの詳細な説明を調べてください。 通常は、問題を修正するために実行できる処置についての説明があります。
  2. ジョブで各パラメーターに合った値を入力したことを確認します。 Web サービス記述のファイル名やエレメントなどのパラメーター値は、大/小文字が区別されます。
  3. 正しいパラメーターの組み合わせを指定したことを確認します。
- CICS JSON アシスタントのバッチ・ジョブを実行すると、戻りコード 1、136、139 のいずれかを受け取ります。 これらの戻りコードは、多くの場合使用可能なストレージが不十分なために JVM で障害が発生したことを意味します。 CICS アシスタントでは、少なくとも 300 MB の JCL 領域サイズが必要です。
  1. 領域サイズを増やすか、領域サイズを `0M` に設定することを検討します。
  2. 領域サイズを制限する可能性があるアクティブな IEFUSI 出口があるかを調べます。
- CICS JSON アシスタントのバッチ・ジョブを実行すると、戻りコード 137 を受け取ります。この戻りコードは、ジョブがタイムアウトになったことを意味します。
  1. ジョブの EXEC ステートメントの **`TIME`** パラメーターを **`TIME=1440`** にコーディングして時間を増やすか、`SYS1.PARMLIB(BPXPRMxx)` メンバーの **`MAXCPUTIME`** 値を増やします。
- DFHJS2LS の実行時に、無効な JSON スキーマやサポートされない JSON スキーマを示す DFHPI9700 から DFHPI9711 の範囲のメッセージを受け取ります。
  1. JSON スキーマが有効であることを確認します。これは、JSON スキーマの妥当性検査と突き合わせてスキーマを確認することで行えます。また、ツール (例えば、json-schema-validator) を使用することもできます。

2. JSON スキーマが DFHJS2LS によってサポートされていることを確認します。詳細については、高水準言語と JSON のスキーマ・マッピングを参照してください。

次のタスク

問題の解決後、JSON アシスタントのバッチ・ジョブを再実行します。

JSON 要求での問題のトラブルシューティング

Web サービスへの入力メッセージとして使用されているか、または JSON を変換するためにリンク可能インターフェースが使用されているときかに関わらず、CICS によって JSON が拒否された場合、JSON が無効であるか、またはスキーマに準拠していない可能性があります。

このタスクについて

Web サービス・リクエスターまたはアプリケーション・プログラムによって提供された JSON に関する問題が CICS で発生した場合、問題の詳細が示されたエラー・メッセージが返されます。 CICS が Web サービス・プロバイダーとして動作する場合、エラー・メッセージはクライアント・アプリケーションに JSON 応答として返されます。詳細については、 696 ページの『クライアントに返されるエラー応答』を参照してください。アプリケーションがリンク可能インターフェースを呼び出して JSON を変換するするときに、エラー・コードが DFHJSON-ERROR コンテナーに返され、詳細なメッセージが DFHJSON-ERRORMSG コンテナーに返されます。詳細については、JSON 変換プログラム・リンク可能インターフェースを参照してください。どちらの場合も、トラブルシューティングの手順は同じで、エラーのタイプに応じます。

手順

- JSON が CICS によって課されている制約事項に準拠していることを確認します。詳細については、JSON Web サービスの制約事項を参照してください。
- JSON 構文解析エラーが発生した場合、JSON が適切な形式であることを確認します。以下のように、エラー・メッセージに問題の詳細が示されます。

  ```
  "Expected a ',' or '}' at character 44 of {"inquireCatalogRequest":
  "myData }
  ```

  ツール (例えば、JSONLint) を使用して JSON 構文を検証できます。
- CICS が JSON をアプリケーション・データにマップしようとしているときに、構造エラーが検出された場合、エラーの詳細を示す DFHPI1007 メッセージが発行されます。 以下に例を挙げます。

  ```
  DFHPI1007 04/19/2013 15:14:42 IYK2ZKE1 00112 JSON to data transformation
  failed because of incorrect input (UNDEFINED_ELEMENT Operation) for
  WEBSERVICE SimpleMappings.
  ```

  DFHJS2LS に指定したスキーマ、または DFHLS2JS が生成したスキーマによって記述された JSON オブジェクトとプロパティーが JSON に含まれていることを確認します。ツールを使用して、JSON がスキーマに準拠していることを検証します。

- エラーが発生していないのに、アプリケーションが一部のフィールドで空のデータを受け取る場合、対応する JSON が指定されていることを確認してください。 CICS は、スキーマによって記述された JSON プロパティーが指定されていないことを検出しません。 JSON 妥当性検査ツールを使用して、以前に記述されたプロパティーを確認できます。
- 内部サーバー・エラーが発生しました。

  ```
  Internal Server Error

  CICS TS: Unhandled Pipeline Error

  Release: 670
  ```

  ブラウザーで、メッセージ DFHSJ1006 を確認します。 JVMServer が存在し、使用不可ではないことを確認します。
- 一部の環境では、CICS は内部処理中に、ラッパー・エレメントを JSON 要求または応答に追加します。これは、アプリケーションでは表示できませんが、エラー・メッセージに含まれる場合があります。 以下に例を挙げます。

  ```
  "Error obtaining parser from data source:Expected a ':' after a key at
  character 25 of {¥"DFHWrapper¥": { l2jb01c¥":"...
  ```

  このような状態では、エラーの原因を特定するときに、DFHWrapper エレメントを無視してください。

次のタスク

JSON を修正した後、アプリケーションを再実行します。

クライアントに返されるエラー応答

CICS JSON ハンドラーで処理中にエラーが発生すると、応答が返されます。

このタスクについて

CICS JSON ハンドラーで処理中にエラーが発生した場合、CICS はそのエラーに関する情報が含まれた応答をクライアントに返します。HTTP 状況コード 500 (内部サーバー・エラー) が返され、エラーの詳細が JSON フォーマットでメッセージ本体に記されます。その内容は生じたエラーのタイプによって異なります。

例

CICS による Axis2 パイプラインの呼び出し前後にエラーが発生すると、メッセージには以下のような情報が含まれます。

```
{
  "exception" : {
     "message" : "An exception has occurred while validating HTTP headers".
     "class" : "com.ibm.cicsts.axis2.Controller"
   }
}
```

処理中のその他の時点でエラーが発生した場合、メッセージには、SOAP 障害と同様の情報と、詳細セクションが含まれます。詳細セクションは、エラーの性質によって異なります。以下のような CICS メッセージが含まれている可能性があります。

```
{
  "Fault": {
     "faultstring": "Conversion from SOAP failed",
     "detail": {
       "CICSFault": "DFHPI1007 02/14/2013 17:51:47 IYK2ZKE1 00185 XML to data
transformation failed because of incorrect input UNDEFINED_ELEMENT startItemRuff)
for WEBSERVICE json_inquireCatalogWrapper."
      }
  }
}
```

JSON アシスタントの戻りコード

JSON アシスタント・バッチ・ジョブを実行している間に障害が発生した場合、障害のタイプを示す戻りコードが示されます。この情報は、ジョブ・ログに含められます。

JSON アシスタント・プログラムの戻りコード

配置エラーが発生した場合、通常 PIPELINE リソースは DISABLED 状態でインストールされ、WEBSERVICE リソースは UNUSABLE 状態でインストールされます。 CICS JSON アシスタントのバッチ・ジョブと関連する情報とエラー・メッセージは、ジョブ・ログにあります。リソースのインストールに関連するエラー・メッセージは、システム・ログにあります。

コード 0、4、8、または 12 はアシスタントによって発行され、それ以外のコードは通常、BPXBATCH、JVM、または IEBGENER によって発行されます。

BPXBATCH によって発行されるコードは 2 つの主要カテゴリーに分類されます。つまり、128 未満のコードはコマンドの失敗を示し、128 以上のコードはシグナルによって処理が終了されたことを示します。 BPXBATCH とその戻りコードについて詳しくは、「z/OS UNIX システム・サービス コマンド解説書」を参照してください。

JSON アシスタントのバッチ・ジョブを実行すると、戻りコード 0、4、8、12 のいずれかを受け取ります。

戻りコード	説明
0	ジョブは正常に完了しました。
4	警告。ジョブは正常に完了しましたが、1 つ以上の警告メッセージが発行されました。
8	入力エラー。ジョブが正常に完了しませんでした。 入力パラメーターの検証中に 1 つ以上のエラー・メッセージが発行されました。
12	エラー。ジョブが正常に完了しませんでした。 実行中に 1 つ以上のエラー・メッセージが発行されました。

JSON アシスタント・バッチ・ジョブの戻りコード

CICS JSON アシスタントのバッチ・ジョブを実行すると、戻りコード 1、136、137、139 のいずれかを受け取ります。

戻りコード	説明
1	JVM で障害が発生しました。通常、これは、使用可能なストレージが不十分であるために発生します。 CICS アシスタントでは、少なくとも 300 MB の JCL 領域サイズが必要です。
136	JVM で障害が発生しました。通常、これは、使用可能なストレージが不十分であるために発生します。 CICS アシスタントでは、少なくとも 300 MB の JCL 領域サイズが必要です。
137	ジョブがタイムアウトになりました。
139	JVM で障害が発生しました。通常、これは、使用可能なストレージが不十分であるために発生します。 CICS アシスタントでは、少なくとも 300 MB の JCL 領域サイズが必要です。

付録 A. JSON 変換プログラム・リンク可能インターフェース・コンテナー

JSON 変換プログラム・リンク可能インターフェースは、JSON との間でデータ変換を行うために呼び出すことができる、CICS が提供するプログラムです。これはアプリケーションの構造化データから JSON データ、または JSON データからアプリケーションの構造化データを作成できます。

リンク可能インターフェースを使用すると、使用できるのは HTTP およびプロバイダー・モードのみである JSON Web サービス・サポートと比べて、WebSphere MQ などのプロトコル経由でも、または JSON データベースからでも、任意のソースから JSON を変換できます。

JSON についての詳細は、JSON Web サービスの概念を参照してください。

JSON 変換プログラム・リンク可能なインターフェースを使用するには、アプリケーション・プログラムが、チャネルを使用して、CICS 提供の DFHJSON プログラムにリンクする必要があります。入力データを DFHJSON に渡し、出力データを DFHJSON から受け取るためにコンテナーが使用されます。これらの詳細はサブトピックで説明します。

CICS では、構造化アプリケーション・データを JSON データとの間で変換するために 2 つのメカニズムがあります。1 つは JVM サーバーを使用し、もう 1 つは使用しません。JVM サーバーを使用する方法では、Java プラットフォームの利点 (使用可能な場合 zEnterprise Application Assist Processor (zAAP) への適格性など) を活用でき、一方 Java 以外の方法では、JVM サーバーを構成する必要がなくなるという利点があります。アプリケーションは、DFHJSON_JVMSERVR コンテナーを使用することで、どちらを使用するかを選択します。コンテナーのコンテンツはデータを変換するための JVM サーバーを特定し、このコンテナーがない場合は、Java 変換サービス以外が使用されることを示します。

このサービスを使用するには、CICS にインストールされている適切な JSONTRANSFRM バンドル・パーツが必要です。これらは、JSON アシスタントを使用して生成されます。詳細については、アプリケーション・データおよび JSON のマッピングと変換を参照してください。

DFHJSON-JSON コンテナー

DFHJSON-JSON は DATATYPE(CHAR) のコンテナーです。

- アプリケーション・データから JSON への変換時: このコンテナーには、アプリケーション・データ入力に対応する、変換プログラムによって作成された JSON が格納されます。
- JSON からアプリケーション・データへの変換時: このコンテナーには、ユーザー・アプリケーションによって提供される変換対象の JSON が格納されます。

© Copyright IBM Corp. 2005, 2018

DFHJSON-DATA コンテナー

DFHJSON-DATA は DATATYPE(BIT) のコンテナーです。

- アプリケーション・データから JSON への変換時: このコンテナーには、ユーザーから提供された、JSON に変換されるアプリケーション・データが格納されます。
- JSON からアプリケーション・データへの変換時: このコンテナーには、JSON 入力に対応する変換データによって作成されるアプリケーション・データが格納されます。

DFHJSON-TRANSFRM コンテナー

DFHJSON-TRANSFRM は DATATYPE(CHAR) のコンテナーです。これには、データ変換用にマッピングが適用される JSONTRANSFRM バンドル・パーツの名前を示す 16 文字が格納されます。

JSONTRANSFRM はバンドル・パーツです。JSON アシスタントを使用することで生成されたバンドルに含まれています。

バンドル・パーツの表示方法について詳しくは、CICS Explorer 製品資料内の『Bundle Parts view』を参照してください。

あるいは、**INQUIRE BUNDLEPART** コマンドを使用することもできます。INQUIRE BUNDLEPART を参照してください。

DFHJSON-JVMSERVR コンテナー

DFHJSON-JVMSERVR は DATATYPE(CHAR) のコンテナーです。これには、ターゲット・プログラムの実行場所となる JVMSERVER の名前を示す 8 文字が含まれます。

コンテナーが提供されない場合、またはコンテナーが存在するが長さがゼロの場合、データ変換は JVM サーバーではなく CICS 内で実行されます。CICS 提供の Axis2 JVM サーバーで変換を実行する場合、このコンテナーに値 DFHAXIS を指定します。

DFHJSON-ERROR コンテナー

DFHJSON-ERROR は DATATYPE(BIT) のコンテナーです。DFHJSON の検証時または実行時に生成されたエラーは、DFHJSON-ERROR コンテナー内のフルワード・バイナリー値として戻されます。

以下のエラーが DFHJSON から戻される可能性があります。

1 DFHJSON-TRANSFRM コンテナーが存在していませんでした。

2 DFHJSON-TRANSFRM コンテナーに提供された JSONTRANSFRM の名前の長さが 16 文字を超えていました。

3 コンテナーのデータ型が間違っていました。DFHJSON-ERRORMSG には、不適切なコンテナーの名前が含まれます。

4 DFHJSON-JVMSERVR コンテナーに提供された JVMSERVER の名前の長さが 8 文字を超えていました。

11 DFHJSON-TRANSFRM コンテナーに指定された JSONTRANSFRM が見つかりませんでした。

12 DFHJSON-TRANSFRM コンテナーに指定された JSONTRANSFRM が ENABLED 状態ではありませんでした。

13 コンテナー DFHJSON-DATA および DFHJSON-JSON のどちらも存在しません。このいずれか 1 つのコンテナーを指定する必要があります。

14 コンテナー DFHJSON-DATA および DFHJSON-JSON の両方が存在します。これらのコンテナーのうち 1 つだけを指定する必要があります。

15 データ変換エラーが発生しました。エラーの詳細が DFHJSON-ERRORMSG に格納されます。

16 DFHJSON-JSON コンテナーに提供された JSON の構文解析中にエラーが発生しました。エラーの詳細が DFHJSON-ERRORMSG に格納されます。

17 変換プログラムの Java 部分でエラーが発生しました。IBM サポート部門に連絡して、支援を求めてください。

20 予期しないエラーが起こりました。IBM サポート部門に連絡して、支援を求めてください。

51 DFHJSON-JVMSERVR コンテナーに指定された JVM サーバーが見つかりませんでした。

52 DFHJSON-JVMSERVR コンテナーに指定された JVM サーバーが ENABLED 状態ではありませんでした。

DFHJSON-ERRORMSG コンテナー

DFHJSON-ERRORMSG は DATATYPE(CHAR) のコンテナーです。このコンテナーは、処理中にエラーが発生した場合に作成されます。これには、人間が解読できる追加のエラー・メッセージが含まれます (例えば JSON パーサーからのエラー・メッセージ)。追加の詳細情報が必要な場合にのみ、このメッセージが設定されます。

このエラー・コードについて詳しくは、 700 ページの『DFHJSON-ERROR コンテナー』を参照してください。

付録 B. Web サービスのサンプル

CICS は、アプリケーションの開発および CICS の構成の開始点として使用できるサンプルのセットを提供します。

サンプルは、以下のように分類されます。

CICS カタログ・マネージャーの実例アプリケーション

CICS カタログ・マネージャー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。

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。

- 『CICS カタログ・マネージャーの実例アプリケーション』

JSON のサンプル

これらの例は、JSON 要求について理解するうえで役立ちます。

- 749 ページの『JSON のサンプル』

CICS カタログ・マネージャーの実例アプリケーション

CICS カタログ・マネージャー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、ユーザーが以下の作業を実行できます。

- 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。

© Copyright IBM Corp. 2005, 2018

この例では、CICS Explorer を使用してアプリケーションをインストールおよびデプロイすることができます。CICS Explorer は、CICS 用の Eclipse ベースのグラフィック・ツール・インターフェースです。

オプション: CICS Explorer を使用しないユーザーは、CEDA トランザクションを使用して、グループ DFH$EXBS および DFH$EXWS に含まれるリソースを修正およびインストールすることができます。

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。

705 ページの図 30 は、ベース・アプリケーションの構造を示しています。

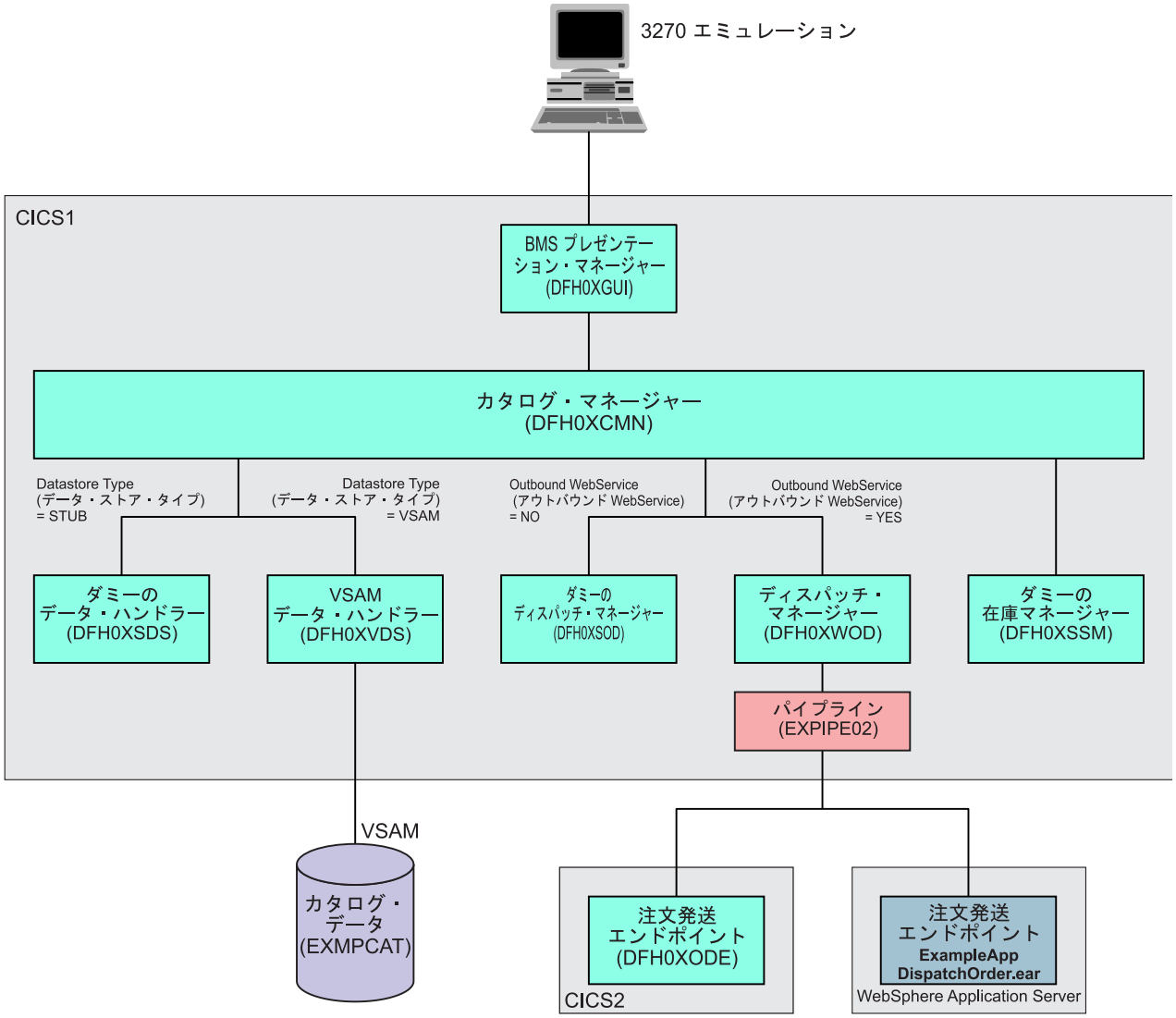


図 *30.* ベース・アプリケーションの構造

ベース・アプリケーションのコンポーネントは次のとおりです。

- BMS プレゼンテーション・マネージャー (DFH0XGUI)。 3270 端末またはエミュレーターをサポートし、メインのカタログ・マネージャー・プログラムと対話します。
- カタログ・マネージャー・プログラム (DFH0XCMN)。実例アプリケーションのコアとなるもので、いくつかのバックエンド・コンポーネントと対話します。
  - データ・ハンドラー・プログラム。カタログ・マネージャー・プログラムとデータ・ストア間のインターフェースを提供します。 ベース・アプリケーションは、このプログラムを 2 種類提供します。 これらのプログラムは、VSAM データ・セットにデータを保管する VSAM データ・ハンドラー・プログラム (DFH0XVDS) と、データを保管せず、その呼び出し元に有効な応答を戻すダミーのデータ・ハンドラー (DFH0XSDS) です。 構成オプションでは、2 つのプログラムのいずれかを選択できます。
  - ディスパッチ・マネージャー・プログラム。注文品を顧客へ発送するためのインターフェースを提供します。 この場合も、構成オプションでは、このプ

ログラムの 2 種類のいずれかを選択することができます。DFHX0WOD は、リモートの注文発送エンドポイントを呼び出す Web サービス・リクエスターで、DFHX0SOD は、その呼び出し元に有効な応答を戻すダミー・プログラムです。

同等の注文発送エンドポイントは 2 つあります。DFH0XODE は CICS サービス・プロバイダー・プログラムで、`ExampleAppDispatchOrderV855.war` は、CICS Liberty JVM サーバーや類似の環境に配置できる Java Web アーカイブ・ファイルです。

– ダミーの在庫マネージャー・プログラム (DFH0XSSM)。呼び出し元に有効な応答を戻すが、その他の動作は行いません。

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。

BMS プレゼンテーション・マネージャーは、次の 2 つの方法で使用できます。

- ベース・アプリケーションの一部として。
- SOAP メッセージを使用してベース・アプリケーションと通信する CICS Web サービス・クライアントとして。

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。

ディスパッチ・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。

注文発送プログラム

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。

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・セットを定義してそこにデータを設定し、2 つの TRANSACTION リソースを作成する必要があります。

VSAM データ・セットの作成および定義

2 つの KSDS VSAM データ・セットを使用して実例アプリケーションを定義し、そこにデータを取り込みます。 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。

手順

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、*hlq*.SDFHINST ライブラリーに置かれます。
   - メンバー DFH$ECNF には、構成データ・セットを生成するための JCL が記述されています。
   - メンバー DFH$ECAT には、カタログ・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、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。

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

ここで、

- *hlq* は、選択した高位修飾子を示します。
- *catname* は、選択した名前を示します。 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。

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. CICS Explorer を使用して、カタログ・ファイルの FILE 定義を作成します。
   a. 「定義 **(Definitions)**」 > 「ファイル定義 **(File Definitions)**」を選択します。 「名前 **(Name)**」列で右クリックし、「新規 **(New)**」をクリックして、新しいファイル定義を作成します。 「リソース・グループ **(Resource Group)**」テキスト・ボックスに EXAMPLE と入力し、「名前 **(Name)**」テキスト・ボックスに EXMPCAT と入力します。「完了 **(Finish)**」をクリックし、FILE 定義を定義します。 または、CICS 提供のグループ DFH$EXBS からファイル定義をコピーすることもできます。
   b. 新規の EXMPCAT ファイルをダブルクリックしてください。 「ファイル定義 (EXMPCAT) CICS 実例アプリケーション (File Definition (EXMPCAT) CICS Example Application)」エディターで「**VSAM**」タブを選択します。 「使用するデータ・セット名 **(Data set name to be used)**」テキスト・ボックスで *hlq*.EXMPLAPP.EXMPCAT と入力します。
   c. 「属性 **(Attributes)**」タブを選択し、以下の属性の操作を「はい **(Yes)**」に設定します。

      追加 (Add)

      参照 (Browse)

      削除 (Delete)

      読み取り (Read)

      更新 (Update)
   d. その他の属性には、すべてデフォルト値を使用します。
5. CICS Explorer を使用して、構成ファイルの FILE 定義を作成します。
   a. 「定義 **(Definitions)**」 > 「ファイル定義 **(File Definitions)**」を選択します。 「名前 **(Name)**」列で右クリックし、「新規 **(New)**」をクリックして、新しいファイル定義を作成します。 「リソース・グループ **(Resource Group)**」テキスト・ボックスに EXAMPLE と入力し、「名前 **(Name)**」テキスト・ボックスに EXMPCONF と入力します。「完了 **(Finish)**」をクリック

し、FILE 定義を定義します。 または、CICS 提供のグループ DFH$EXBS からファイル定義をコピーすることもできます。

b. 新規の EXMPCONF ファイルをダブルクリックしてください。 「ファイル定義 (EXMPCONF) CICS 実例アプリケーション (File Definition (EXMPCONF) CICS Example Application)」ウィンドウで「**VSAM**」タブを選択します。 「使用するデータ・セット名 **(Data set name to be used)**」テキスト・ボックスで *hlq*.EXMPLAPP.EXMPCONF と入力します。

c. 「属性 **(Attributes)**」タブを選択し、以下の属性の操作を「はい **(Yes)**」に設定します。

追加 (Add)

参照 (Browse)

削除 (Delete)

読み取り (Read)

更新 (Update)

d. その他の属性には、すべてデフォルト値を使用します。

タスクの結果

データ・セットにデータが取り込まれ、カタログ・ファイルおよび構成ファイルの FILE 定義が作成されて、インストールの準備が整いました。

オプション: CICS Explorer を使用しないユーザーは、CEDA トランザクションを使用して、グループ DFH$EXBS および DFH$EXWS に含まれるリソースを修正およびインストールすることができます。

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 ECFG の 2 つのトランザクションで構成されます。 3 番目のトランザクションである ECLI は、CICS Web サービス・クライアントで使用されます。

このタスクについて

オプション: CICS Explorer を使用しないユーザーは、CEDA トランザクションを使用して、グループ DFH$EXBS および DFH$EXWS に含まれるリソースを修正およびインストールすることができます。

手順

1. CICS Explorer を使用して、以下のトランザクション用の TRANSACTION 定義を作成します。 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、別の名前を使用できます。

EGUI

a. 「リソース・グループ定義」ビューで DFH$EXBS を右クリックして、トランザクション EGUI 用の定義を CICS 付属のグループ DFH$EXBS からコピーします。

b. 「新規 **(New)**」 > 「トランザクション定義 **(Transaction Definition)**」を選択します。

c. 「名前」テキスト・ボックスに EGUI と入力し、「リソース**/CSD** グループ **(Resource/CSD Group)**」テキスト・ボックスに名前を指定し、「プログラム名」テキスト・ボックスに DFH0XGUI と入力します。

d. 「完了 **(Finish)**」をクリックし、EGUI トランザクション定義を作成します。

その他の属性には、すべてデフォルト値を使用します。

ECFG

a. 「リソース・グループ定義」ウィンドウで DFH$EXBS を右クリックして、トランザクション ECFG 用の定義を CICS 付属のグループ DFH$EXBS からコピーします。

b. 「新規 **(New)**」 > 「トランザクション定義 **(Transaction Definition)**」を選択します。

c. 「名前」テキスト・ボックスに ECFG と入力し、「リソース**/CSD** グループ **(Resource/CSD Group)**」テキスト・ボックスに名前を指定し、「プログラム名」テキスト・ボックスに DFH0XCFG と入力します。

d. 「完了 **(Finish)**」をクリックし、ECFG トランザクション定義を作成します。

その他の属性には、すべてデフォルト値を使用します。

ECLI (オプション**)**

a. 「リソース・グループ定義」ビューで DFH$EXWS を右クリックして、トランザクション EGUI 用の定義を CICS 付属のグループ DFH$EXWS からコピーします。

b. 「新規 **(New)**」 > 「トランザクション定義 **(Transaction Definition)**」を選択します。

c. 「名前」テキスト・ボックスに ECLI と入力し、「リソース**/CSD** グループ **(Resource/CSD Group)**」テキスト・ボックスに名前を指定し、「プログラム名」テキスト・ボックスに DFH0XCUI と入力します。

d. 「完了 **(Finish)**」をクリックし、ECLI トランザクション定義を作成します。

その他の属性には、すべてデフォルト値を使用します。

2. オプション: プログラムの自動インストールを使用したくない場合は、ベース・アプリケーション・プログラム用の PROGRAM 定義および BMS マップ用の MAPSET 定義を CICS 付属のグループ DFH$EXBS からコピーします。

a. メンバー DFH0XS1、DFH0XS2、および DFH0XS3 で BMS マップの MAPSET リソース定義をコピーします。 各メンバーに含まれる内容について詳しくは、 738 ページの『ベース・アプリケーションのコンポーネント』を参照してください。

b. 以下の COBOL プログラム用の PROGRAM リソース定義をコピーします。

表 16. ベース・アプリケーションの *COBOL* ソースが含まれる *SDFHSAMP* メンバー

メンバー名	説明
DFH0XCFG	VSAM 構成ファイルを読み取って更新するために、トランザクション ECFG によって呼び出されるプログラム。
DFH0XCMN	カタログ・アプリケーションのコントローラー・プログラム。すべての要求がコントローラー・プログラムをパススルーします。
DFH0XGUI	端末ユーザーへの BMS マップの送信および端末ユーザーからのマップの受信を管理するために、トランザクション EGUI によって呼び出されるプログラム。 このプログラムはプログラム DFH0XCMN にリンクします。
DFH0XODE	注文発送 Web サービスの 2 つのバージョンのエンドポイントのいずれか。 これは、CICS で稼働するバージョンです。このプログラムは、戻りの COMMAREA でテキスト「Order in dispatch」を設定します。
DFH0XSDS	VSAM カタログ・ファイルがセットアップされていない場合に、アプリケーションが操作できるスタブ化 版またはダミー版のデータ・ストア・プログラム。 DFH0XSDS は、VSAM ファイルに保管されたデータではなく、プログラムで定義されたデータを使用します。
DFH0XSOD	スタブ化版の注文発送プログラム。このプログラムは、COMMAREA 内の戻りコードを 0 に設定して、呼び出し元に戻します。 DFH0XSOD は、アウトバウンド Web サービスが必要ないときに使用されます。
DFH0XSSM	スタブ化版の在庫マネージャー (補充) プログラム。 DFH0XSSM は、COMMAREA 内の戻りコードを 0 に設定して、呼び出し元に戻します。
DFH0XVDS	VSAM 版のデータ・ストア・プログラム。 DFH0XVDS は VSAM ファイルにアクセスして、カタログの読み取りと更新を実行します。
DFH0XWOD	Web サービス版の注文発送プログラム。 DFH0XWOD は EXEC CICS INVOKE WEBSERVICE を発行して、注文発送プログラムへのアウトバウンド Web サービス呼び出しを行います。

その他の属性には、すべてデフォルト値を使用します。

c. オプション: DFH0XCUI 用の PROGRAM 定義を CICS 付属のグループ DFH$EXWS からコピーします。 その他の属性には、すべてデフォルト値を使用します。 Web サービス・クライアントを開始するトランザクション ECLI を使用する場合、このプログラムは必須です。

```
DIS G(DFH$EXWS)
ENTER COMMANDS
NAME     TYPE         GROUP
DFH0XCUI PROGRAM      DFH£EXWS
ECLI     TRANSACTION  DFH£EXWS
EXMPPORT TCPIPSERVICE DFH£EXWS
EXPIPE01 PIPELINE     DFH£EXWS
EXPIPE02 PIPELINE     DFH£EXWS
```

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。

手順

「リソース・グループ定義 (Resource Group Definitions)」ウィンドウでリソース・グループを右クリックします。 「インストール **(Install)**」を選択します。 CICSplex が正しいものであり、ターゲット領域を選択していることを確認してから、「**OK**」をクリックします。

タスクの結果

これで、RDO がインストールされ、アプリケーションを使用する準備が整いました。

オプション: CICS Explorer を使用しないユーザーは、CEDA トランザクションを使用して、グループ DFH$EXBS および DFH$EXWS に含まれるリソースを修正およびインストールすることができます。

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。

始める前に

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。

このタスクについて

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。

手順

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。 以下の画面が CICS によって表示されます。

```
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)
データ・ストア・スタブ・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・プログラムを使用する場合は、VSAM を指定します。

Outbound WebService (アウトバウンド WebService)
注文発送機能にリモート Web サービスを使用する場合、つまり、カタログ・マネージャー・プログラムを注文発送 Web サービス・プログラムにリンクする場合は、YES を指定します。

注文発送機能にスタブ・プログラムを使用する場合、つまり、カタログ・マネージャー・プログラムを注文発送スタブ・プログラムにリンクする場合は、NO を指定します。

Catalog Manager (カタログ・マネージャー)
カタログ・マネージャーの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、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)
「**Outbound WebService (**アウトバウンド **WebService)**」フィールドに NO を指定した場合は、注文発送スタブの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)
在庫マネージャーの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)
CICS Web サービス・クライアントを使用する場合は、実例アプリケーションが Web サービスとして配置されているシステムの IP アドレスとポートを指定します。

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、「アウトバウンド **Web** サービス」を `http://myserver:myport/exampleApp/dispatchOrder` に指定します。ここで、*myserver* および *myport* は、使用している CICS サーバーのアドレスとポートです。

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、BMS インターフェースを使用して実行できます。

手順

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・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
```

このメニューのオプションを使用して、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 を押して、「品目のリスト表示 (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. 「注文 **(Order)**」列に / と入力し、Enter 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. 「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. 「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. 「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. F3 を押してアプリケーションを終了します。

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、2 つの Java バージョンの Web サーバー・フロントエンド・クライアント、および Java バージョンと COBOL バージョンの Web サービス・エンドポイントを、注文発送プログラム・コンポーネントに提供します。

2 つのバージョンの Web クライアント・フロントエンドおよび 1 つのバージョンの Web サービス・エンドポイントが、最新バージョンの WebSphere Application Server または最新の CICS TS Liberty JVM サーバーによって提供される Java EE 6 Web プロファイル環境で実行される Java Web アーカイブ・ファイル (WAR) として提供されます。2 番目のバージョンの Web サービス・エンドポイントは、CICS サービス・プロバイダー・アプリケーション・プログラム (DFH0XODE) として提供されます。

ファイル	説明
ExampleAppClientV855.war	カタログ・マネージャーへの Web サービス・フロントエンド・クライアント

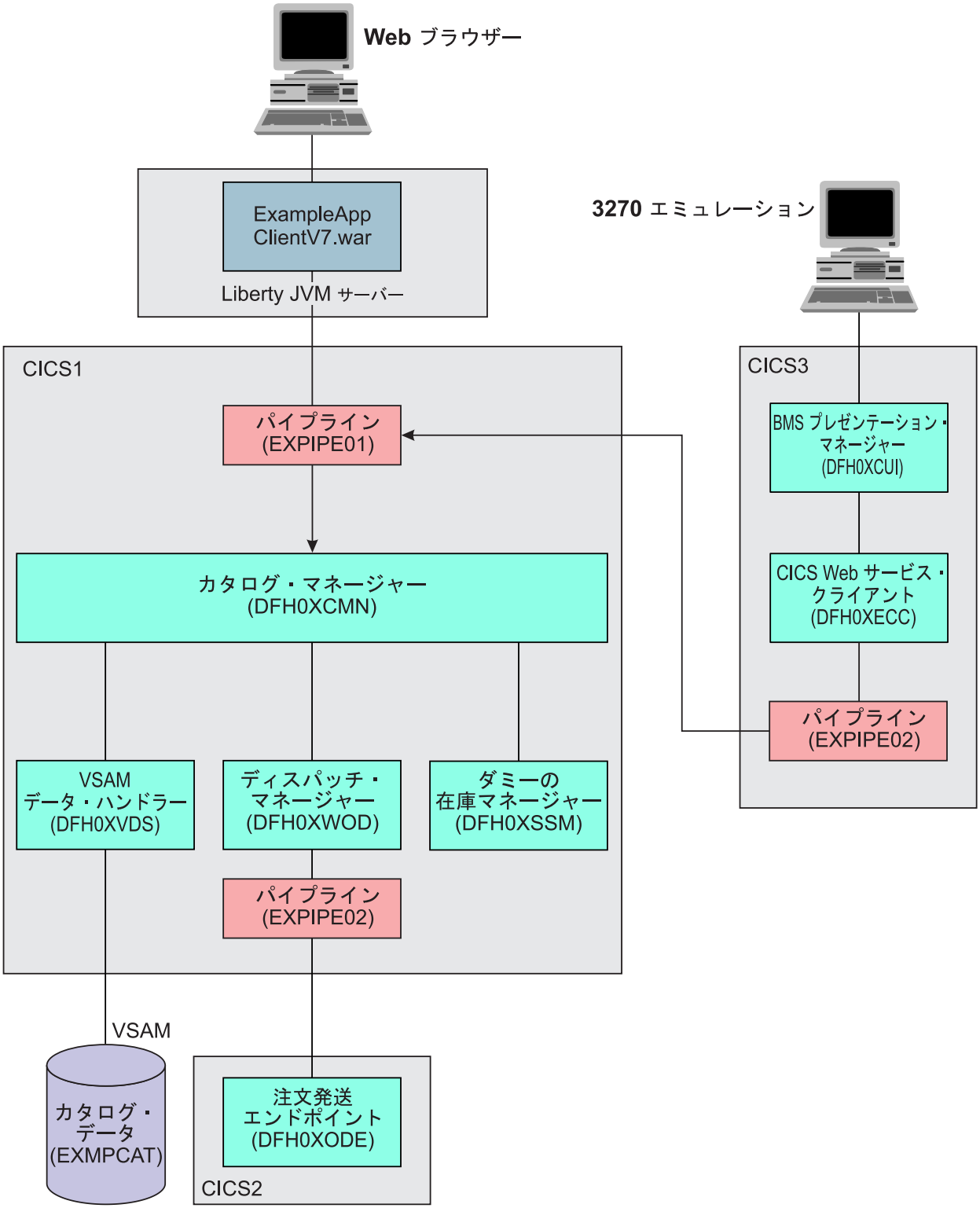
ファイル	説明
ExampleAppWrapperClientV855.war	Web サービス・ラッパーへの Web サービス・フロントエンド・クライアント
ExampleAppDispatchOrderV855.war	注文発送 Web サービス・プロバイダー・アプリケーション

これらの WAR は、動的 Web プロジェクトからエクスポートされました。WAR ファイルをデプロイする方法については、JVM サーバーへのアプリケーションのデプロイを参照してください。

Liberty JVM サーバーで、JAX-WS フィーチャーを有効にする必要があります。それには、例えば以下のコードを Liberty サーバー構成ファイル `server.xml` に追加します。

```
<feature>jaxws-2.2</feature>
```

図 1 は、あるバージョンの Web クライアント・フロントエンドと CICS サービス・プロバイダーを注文発送 Web サービス・エンドポイントとして持つサンプル・アプリケーションの構成を示しています。これには、 CICS システム上の Web サービス・クライアントも含まれます。

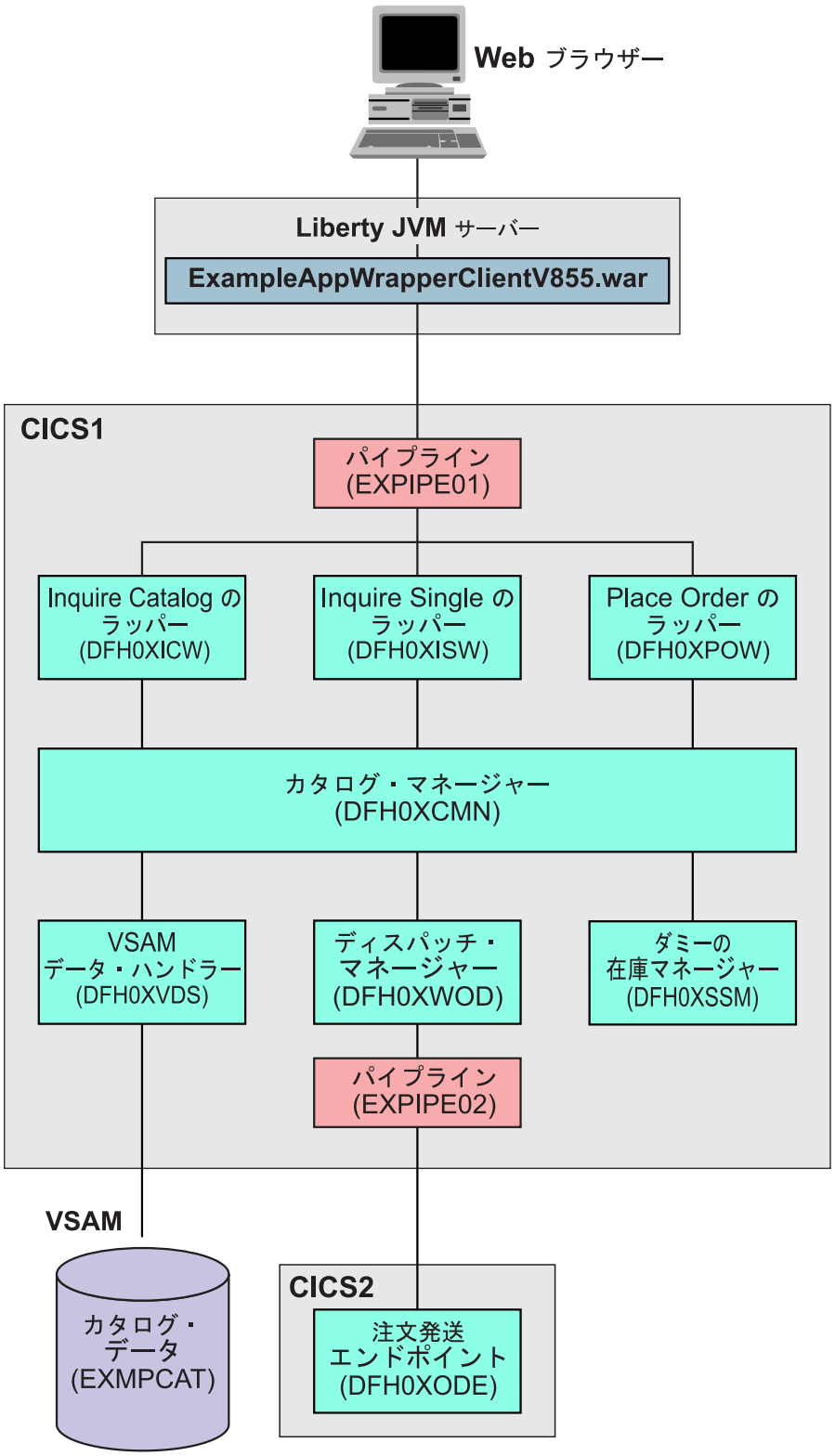


この構成では、アプリケーションは 2 つの異なるクライアントからアクセスされます。

- Liberty JVM サーバーに接続された Web ブラウザー・クライアント。ここに `ExampleAppClientV855.War` が配置されます。

- CICS Web サービス・クライアント DFH0XECC。 このクライアントはベース・アプリケーションと同じ BMS プレゼンテーション論理を使用しますが、DFH0XGUI の代わりに DFH0XCUI モジュールを使用します。

図 2 は、CICS サービス・プロバイダーを注文発送 Web サービス・エンドポイントとして持つ、別のバージョンの Web クライアント・フロントエンドを示しています。



この構成では、Web ブラウザー・クライアントは Liberty JVM サーバーに接続されます。ここに `ExampleAppWrapperClientV855.war` が配置されます。CICS では、3 つのラッパー・アプリケーション (Inquire Catalog、Inquire Single、および Place

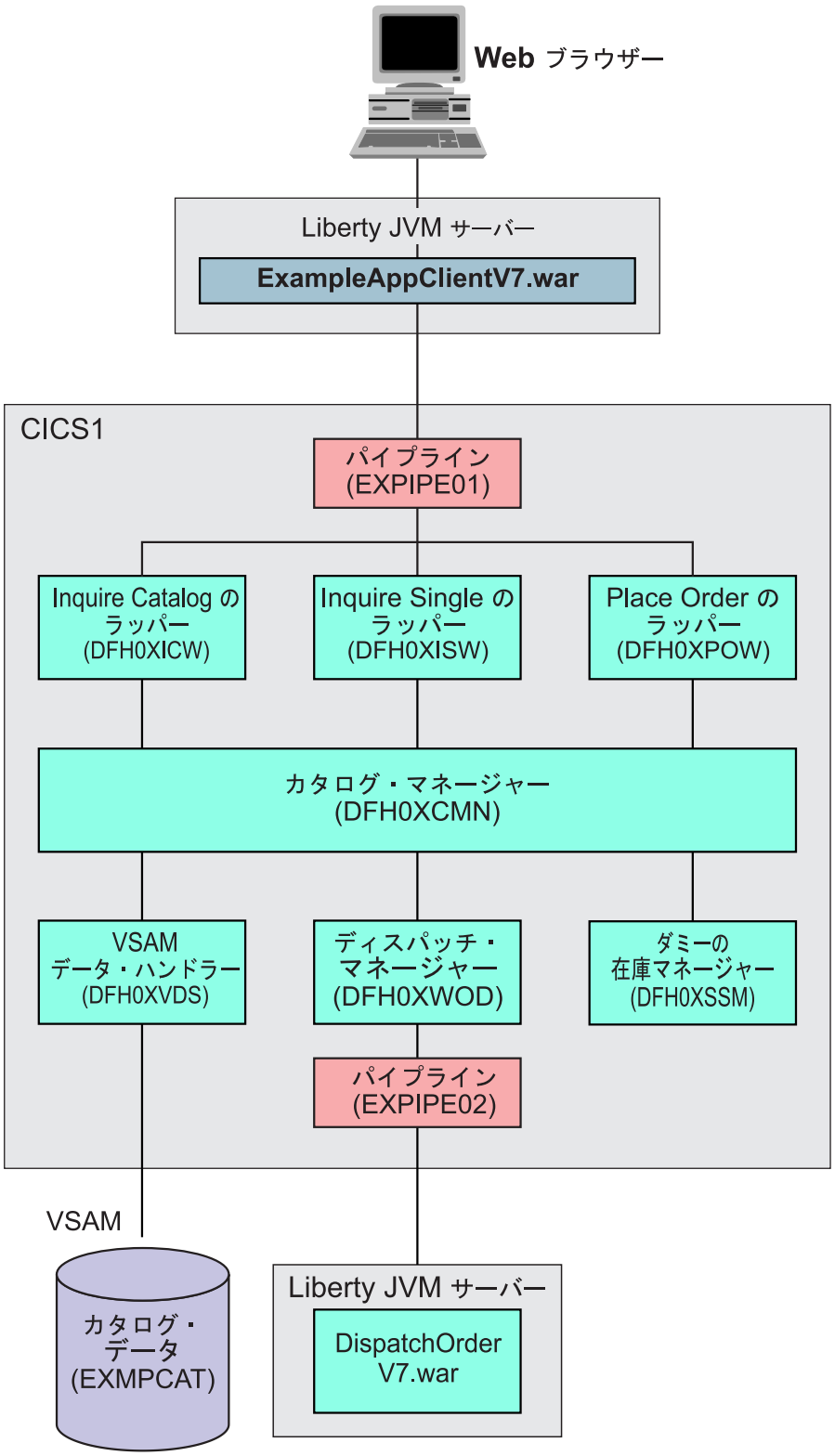
Order の機能用) がサービス・プロバイダー・アプリケーションとして配置されます。これらのアプリケーションは、ベース・アプリケーションに順にリンクします。

CICS システム上の発送マネージャーがこのエンドポイントを呼び出すようにするには、ECFG 構成トランザクションを使用して、以下の構成を変更する必要があります。

- Outbound WebService? を YES に
- アウトバウンド Web サービス URI を、注文発送エンドポイントが配置されている場所の URI (例えば `http://cics2:8080/exampleApp/dispatchOrder`) に変更

実例アプリケーションのセットアップについて詳しくは、サンプル・アプリケーションの構成を参照してください。

図 3 は、Liberty JVM サーバー上に Web クライアント・フロントエンドと注文発送 Web サービス・エンドポイントの両方があるサンプル・アプリケーションの構成を示しています。



この構成では、Web ブラウザー・クライアントは ExampleAppClientV855.war が配置されている Liberty JVM サーバーに接続されます。注文発送 Web サービス・エンドポイント ExampleAppDispatchOrderV855.war が Liberty JVM サーバーにインストールされました。

CICS システム上の発送マネージャーがこのエンドポイントを呼び出すようにするには、ECFG 構成トランザクションを使用して、以下の構成を変更する必要があります。

- Outbound WebService? を YES に
- アウトバウンド Web サービス URI を、注文発送エンドポイントが配置されている場所の URI (例えば `http://mylibertyserver:9080/ExampleAppDispatchOrderV855/DispatchOrder`) に変更

実例アプリケーションのセットアップについて詳しくは、サンプル・アプリケーションの構成を参照してください。

コード・ページ・サポートの構成

実例アプリケーションは、提供された状態では 2 つのコード化文字セットを使用します。 2 つの文字セット間でデータ変換を行えるようシステムを構成する必要があります。

このタスクについて

この実例アプリケーションで使用されるコード化文字セットは、以下のとおりです。

037 EBCDIC グループ 1: 米国、カナダ (z/OS)、オランダ、ポルトガル、ブラジル、オーストラリア、ニュージーランド)

1208 UTF-8 レベル 3

手順

ご使用の z/OS システムの変換イメージに次のステートメントを追加します。

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

詳細については、z/OS による Unicode データ変換を参照してください。

Web サービス・クライアントおよびラッパー・プログラムの定義

プログラムの自動インストールを使用しない場合は、Web サービス・クライアントおよびラッパー・プログラムのリソース定義を定義する必要があります。

このタスクについて

オプション: CICS Explorer を使用しないユーザーは、CEDA トランザクションを使用して、グループ DFH$EXBS および DFH$EXWS に含まれるリソースを修正およびインストールすることができます。

手順

CICS Explorer を使用して、ラッパー・プログラムの PROGRAM リソース定義を定義します。これを行うには、「定義 **(Definitions)**」 > 「プログラム定義 **(Program Definitions)**」を選択します。 「プログラム定義 (Program Definitions)」ビューで右クリックし、「新規 **(New)**」を選択して、新しいプログラム定義を作成します。「**CSD** グループ」テキスト・ボックスに`CSD グループ`を入力

し、「名前」テキスト・ボックスにプログラム名を入力します。「完了 **(Finish)**」をクリックし、PROGRAM 定義を定義します。 以下の COBOL プログラム用の定義を作成します。

表 *17.* ラッパー・モジュールの *COBOL* ソース・コードが含まれる *SDFHSAMP* メンバー

メンバー名	説明
DFH0XECC	Web サービス・クライアント・プログラム
DFH0XICW	`inquireCatalog` サービスのラッパー・プログラム。
DFH0XISW	`inquireSingle` サービスのラッパー・プログラム。
DFH0XPOW	`purchaseOrder` サービスのラッパー・プログラム。

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 z/OS UNIX ディレクトリーを作成し、必要な CICS リソースを作成しておく必要があります。

z/OS UNIX ディレクトリー**:**

実例アプリケーションの Web サービス・サポートでは、z/OS UNIX 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 WEBSERVICE 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、複数の PIPELINE 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 WEBSERVICE 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。

パイプライン定義の作成**:**

パイプラインの完全な定義は、PIPELINE リソースと PIPELINE 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。

このタスクについて

オプション: CICS Explorer を使用しないユーザーは、CEDA トランザクションを使用して、グループ DFH$EXBS および DFH$EXWS に含まれるリソースを修正およびインストールすることができます。

実例アプリケーションでは、提供される 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 Web サービスが 1 つのパイプラインを共用できるため、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 つのパイプラインを、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 番目のパイプラインを定義する必要があります。

Java ベースのパイプラインを使用する場合は、ステップ 1b で、サンプル・サービス・プロバイダー構成ファイル `basicsoap11provider.xml` の代わりに、`basicsoap11javaprovider.xml` を指定する必要があります。 また、ステップ 2b で、サンプル・サービス・リクエスター構成ファイル `basicsoap11requester.xml` の代わりに、`basicsoap11javarequester.xml` を指定する必要があります。 サンプル構成ファイルについて詳しくは、パイプライン構成ファイル を参照してください。また、Java ベースのパイプラインで Axis2 アプリケーション・ハンドラーを使用する場合は、ステップ 1a で `EXPIPE01` を `EXPIPE03` に置き換え、ステップ 2a で `EXPIPE02` を `EXPIPE04` に置き換える必要があります。

手順

1. CICS Explorer を使用して、サービス・プロバイダーのパイプライン定義を作成します。
   a. CICS Explorer を使用して、ラッパー・プログラムの PIPELINE 定義を作成します。これを行うには、「定義 **(Definitions)**」 > 「パイプライン定義 **(Pipeline Definitions)**」を選択します。 「パイプライン定義 (Pipeline Definitions)」ビューで右クリックし、「新規 **(New)**」を選択して、新しいパイプライン定義を作成します。 「リソース・グループ **(Resource Group)**」テキスト・ボックスに `DFH$EXWS` と入力し、「名前 **(Name)**」テキスト・ボックスに `EXPIPE01` と入力します。 「完了 **(Finish)**」をクリックし、PIPELINE 定義を作成します。 または、CICS 提供のグループ DFH$EXWS から PIPELINE 定義をコピーすることもできます。 「リソース・グループ定義 (Resource Group Definition)」ビューで DFH$EXWS を右クリックし、「新規 **(New)**」 > 「パイプライン定義 **(Pipeline Definition)**」を選択します。
   b. PIPELINE 定義をダブルクリックし、「パイプライン定義 (EXPIPE01) (Pipeline Definition (EXPIPE01))」エディターから「属性 **(Attributes)**」タブを選択します。「詳細 **(Details)**」で、「構成ファイル **(Configuration File)**」をサンプル・ファイルの場所 */usr/lpp/cicsts/*samples/pipelines/basicsoap11provider.xml (*/usr/lpp/cicsts* はご使用のディレクトリー上のファイルへのパス) に設定する必要があります。また、「シェルフ **(Shelf)**」は `/var/cicsts/`、「状況 **(Status)**」は `ENABLED`、「**WS** ディレクトリー **(WS Directory)**」は */usr/lpp/cicsts/*samples/webservices/wsbind/provider/ でなければなりません。

      z/OS UNIX の項目では大/小文字が区別され、デフォルトの CICS z/OS UNIX インストール・ルートが `/usr/lpp/cicsts` であることを想定します。

2. CICS Explorer 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. CICS Explorer を使用して、ラッパー・プログラムの PIPELINE 定義を作成します。これを行うには、「定義 **(Definitions)**」 > 「パイプライン定義 **(Pipeline Definitions)**」を選択します。 「パイプライン定義 (Pipeline Definitions)」ビューで右クリックし、「新規 **(New)**」を選択して、新しいパイプライン定義を作成します。 「リソース・グループ **(Resource Group)**」テキスト・ボックスに `DFH$EXWS` と入力し、「名前 **(Name)**」テキスト・ボックスに `EXPIPE02` と入力します。 「完了 **(Finish)**」をクリックし、PIPELINE 定義を作成します。 または、CICS 提供のグループ DFH$EXWS から PIPELINE 定義をコピーすることもできます。
   b. PIPELINE 定義をダブルクリックし、「パイプライン定義 (EXPIPE02) (Pipeline Definition (EXPIPE02))」エディターから「属性 **(Attributes)**」タブを選択します。「詳細 **(Details)**」で、「構成ファイル **(Configuration File)**」をサンプル・ファイルの場所 `/usr/lpp/cicsts/samples/pipelines/basicsoap11requester.xml` (`/usr/lpp/cicsts` はご使用のディレクトリー上のファイルへのパス) に設定する必要があります。「シェルフ **(Shelf)**」は `/var/cicsts/`、「状況 **(Status)**」は `ENABLED`、「**WS** ディレクトリー **(WS Directory)**」は `/usr/lpp/cicsts/samples/webservices/wsbind/requester/` でなければなりません。

TCP/IP サービスの作成**:**

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、TCP/IP サービス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。

手順

インバウンド HTTP 要求を処理するには、CICS Explorer を使用して TCPIPSERVICE 定義を作成します。

1. 「定義 **(Definitions)**」 > 「**TCP/IP** サービス定義 **(TCP/IP Service Definitions)**」を選択して、TCPIPSERVICE 定義を作成します。 「TCP/IP サービス定義 (TCP/IP Service Definitions)」ビューで右クリックし、「新規 **(New)**」を選択して、新しい定義を作成します。「リソース・グループ **(Resource Group)**」テキスト・ボックスに `DFH$EXWS` と入力し、「名前 **(Name)**」テキスト・ボックスに `EXMPPORT` と入力します。 ポート番号を指定する必要があります。CICS システムで使用されていないポートの番号を入力します。 「完了 **(Finish)**」をクリックし、TCPIPSERVICE 定義を作成します。
2. TCPIPSERVICE 定義をダブルクリックします。 「TCP/IP サービス定義 (EXMPPORT) (TCP/IP Service Definition (EXMPPORT))」エディターの「属性 **(Attributes)**」タブで、以下の属性を設定します。

   「**URM (Urm)**」は `DFHWBAAX` にする必要があります。

   「プロトコル **(Protocol)**」は `HTTP` にする必要があります。

   「トランザクション **(Transaction)**」は `CWXN` にする必要があります。
3. その他の属性には、すべてデフォルト値を使用します。

オプション: CICS Explorer を使用しないユーザーは、CEDA トランザクションを使用して、グループ DFH$EXBS および DFH$EXWS に含まれるリソースを修正およびインストールすることができます。

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、着信要求を正しいパイプラインおよび Web サービスに送信する URIMAP リソースが必要です。オンライン・リソース定義 (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、パイプライン・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。

このタスクについて

オプション: CICS Explorer を使用しないユーザーは、CEDA トランザクションを使用して、グループ DFH$EXBS および DFH$EXWS に含まれるリソースを修正およびインストールすることができます。

手順

1. CICS Explorer を使用して PIPELINE リソースをインストールします。
   a. 「定義 **(Definitions)**」 > 「パイプライン定義 **(Pipeline Definitions)**」を選択します。 「パイプライン定義 (Pipeline Definitions)」ビューで EXPIPE01 PIPELINE 定義を右クリックし、「インストール **(Install)**」を選択します。 ターゲットの CICS 領域をチェック・ボックスで選択します。「**OK**」をクリックし、PIPELINE をインストールします。

      注**:** 724 ページの『パイプライン定義の作成』で Java ベースのパイプライン定義を作成した場合は、パイプライン定義ビューで EXPIPE03 PIPELINE 定義を右クリックします。
   b. EXPIPE02 PIPELINE 定義、または Java ベース・パイプライン用の EXPIPE04 で、このプロセスを繰り返します。

      各 PIPELINE リソースをインストールすると、CICS は、PIPELINE 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 WEBSERVICE 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 リソースおよび URIMAP リソースをインストールします (それらが存在しなかった場合)。

      URIMAP リソースは、WEBSERVICE リソースを特定の URI に関連付けるための情報を CICS に提供します。 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。

      WSDL ファイルまたは WSDL アーカイブ・ファイルがピックアップ・ディレクトリーにコピーされている場合、2 番目のオプション URIMAP リソースがインストールされます。 この URIMAP リソースは、WSDL アーカイブ・ファイルまたは WSDL 文書を特定の URI に関連付けるための情報

を CICS に提供します。これにより外部リクエスターは、その URI を使用して WSDL アーカイブ・ファイルまたは WSDL 文書を見つけることができます。

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。

2. PIPELINE リソースをインストール済みの場合は、**PERFORM PIPELINE SCAN** コマンドを使用して、PIPELINE 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 リソースをインストールすると、プロバイダー・パイプライン用の以下の WEBSERVICE リソースと、それに関連した URIMAP リソースがシステムにインストールされます。

   ```
   dispatchOrder
   dispatchOrderEndpoint
   inquireCatalog
   inquireCatalogClient
   inquireCatalogWrapper
   inquireSingle
   inquireSingleClient
   inquireSingleWrapper
   placeOrder
   placeOrderClient
   placeOrderWrapper
   ```

   WEBSERVICE リソースの名前は、WEBSERVICE バインディング・ファイルの名前から得られます。URIMAP リソースの名前は動的に生成されます。 追加 URIMAP が、パイプラインのピックアップ・ディレクトリーに存在する各 WSDL 文書について生成されます。このリソースを表示するには、「操作 **(Operations)**」 > 「**Web** サービス **(Web Services)**」を選択して「Web サービス (Web Services)」ビューを開きます。 WEBSERVICE リソースを右クリックし、「関連を開く **(Open Related)**」 > 「**URI** マップ **(URI Map)**」を選択します。

   CICS Explorer ビューに、PIPELINE リソースと URIMAP リソースの名前、および各 Web サービスに関連したターゲット・プログラムの名前が表示されます。 この例では、WEBSERVICE リソースはアウトバウンド要求を対象にしているため、WEBSERVICE(dispatchOrder) に URIMAP もターゲット・プログラムもありません。

   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。

オンライン・リソース定義 **(RDO)** による **WEBSERVICE** リソースの作成**:**

WEBSERVICE リソースは、パイプライン・スキャン機能を使用してインストールする代わりに、オンライン・リソース定義 (RDO) を使用して作成およびインストールすることができます。

始める前に

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。これによって、ピックアップ・ディレクトリーのパイプライン・スキャン中に、WEBSERVICE リソースおよび URIMAP リソースが動的にインストールされないことが保証されます。 または、PIPELINE 内の WSDIR に値が指定されていないことを確認します。ただし、WSDIR に値を指定しないと、ピックアップ・ディレクトリーのパイプライン・スキャンが実行されません。 そのため、すべての WEBSERVICE リソースおよび URIMAP リソースを、RDO を使用して作成およびインストールする必要があります。

手順

1. CICS Explorer を使用して、実例アプリケーションの INQUIRE CATALOG 機能の WEBSERVICE 定義を作成します。
   a. CICS Explorer を使用して、WEBSERVICE 定義を作成します。これを行うには、「定義 **(Definitions)**」 > 「**Web** サービス定義 **(Web Service Definition)**」を選択します。
   b. 「Web サービス定義 (Web Service Definitions)」ビューで右クリックし、「新規 **(New)**」を選択して、新しい WEBSERVICE 定義を作成します。
   c. 「リソース・グループ **(Resource Group)**」テキスト・ボックスに `DFH$EXWS`、「名前 **(Name)**」テキスト・ボックスに `EXINQCWS`、「パイプライン **(Pipeline)**」テキスト・ボックスに `EXPIPE01` (Java ベースのパイプラインの場合は `EXPIPE03`) と入力します。 WEBSERVICE 定義を作成するには、その前に WSBind 属性を入力する必要があります。 「**WSBind** ファイル **(WSBind File)**」テキスト・ボックスに `/usr/lpp/cicsts/samples/webservices/wsbind/provider/inquireCatalog.wsbind` と入力します。
   d. 「完了 **(Finish)**」をクリックし、WEBSERVICE 定義を作成します。
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 または EXPIPE03	`/usr/lpp/cicsts/samples/webservices/wsbind/provider/inquireSingle.wsbind`
PLACE ORDER	EXORDRWS	EXPIPE01 または EXPIPE03	`/usr/lpp/cicsts/samples/webservices/wsbind/provider/placeOrder.wsbind`
DISPATCH STOCK	EXODRQWS	EXPIPE02 または EXPIPE04	`/usr/lpp/cicsts/samples/webservices/wsbind/requester/dispatchOrder.wsbind`
DISPATCH STOCK endpoint (オプショナル)	EXODEPWS	EXPIPE01 または EXPIPE03	`/usr/lpp/cicsts/samples/webservices/wsbind/provider/dispatchOrderEndpoint.wsbind`

オンライン・リソース定義 **(RDO)** による **URIMAP** リソースの作成**:**

URIMAP リソースは、パイプライン・スキャン機能を使用してインストールする代わりに、オンライン・リソース定義 (RDO) を使用して作成およびインストールすることができます。

始める前に

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。これによって、ピックアップ・ディレクトリーのパイプライン・スキャン中に、WEBSERVICE リソースおよび URIMAP リソースが動的にインストールされないことが保証されます。 または、PIPELINE 内の WSDIR に値が指定されていないことを確認します。ただし、WSDIR に値を指定しないと、ピックアップ・ディレクトリーのパイプライン・スキャンが実行されません。 そのため、すべての WEBSERVICE リソースおよび URIMAP リソースを、RDO を使用して作成およびインストールする必要があります。

手順

1. CICS Explorer を使用して、実例アプリケーションの INQUIRE CATALOG 機能の URIMAP 定義を作成します。
   a. CICS Explorer で URIMAP 定義を作成します。これを行うには、「定義 **(Definitions)**」 > 「**URI** マップ定義 **(URI Map Definition)**」を選択します。
   b. 「URI マップ定義 (URI Map Definition)」ビューで右クリックし、「新規 **(New)**」を選択して、新しい URIMAP 定義を作成します。
   c. 「名前 **(Name)**」テキスト・ボックスに `INQCURI` と入力し、「ホスト **(Host)**」テキスト・ボックスに `*` と入力します。 URIMAP 定義を作成するには、その前に **Path** 属性を入力する必要があります。 「パス **(Path)**」テキスト・ボックスで `/exampleApp/inquireCatalog` と入力します。 「使用法 **(Usage)**」は「パイプライン **(Pipeline)**」に設定する必要があります。 PIPELINE リソースは `EXPIPE01` (Java ベースのパイプラインでは `EXPIPE03`) です。
   d. 「完了 **(Finish)**」をクリックし、URIMAP 定義を定義します。
   e. 新しい URIMAP リソースをダブルクリックし、「エディター (Editor)」を開きます。「エディター (Editor)」の「属性 **(Attributes)**」タブで、「**Web** サービス **(Web Service)**」属性を `EXINQCWS` に「**TCP/IP** サービス **(TCP/IP Service)**」を `SOAPPORT` に設定します。
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

機能	**URIMAP** 名
DISPATCH STOCK endpoint (オプショナル)	ODEPURI

3. 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
DISPATCH STOCK endpoint (オプショナル)	ODEPURI	/exampleApp/dispatchOrder	EXODEPWS

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。

手順

「リソース・グループ定義 (Resource Group Definitions)」ウィンドウでリソース・グループを右クリックします。 「インストール **(Install)**」を選択します。 CICSplex が正しいものであり、ターゲット領域を選択していることを確認してから、「**OK**」をクリックします。

タスクの結果

これで、RDO がインストールされ、アプリケーションを使用する準備が整いました。

オプション: CICS Explorer を使用しないユーザーは、CEDA トランザクションを使用して、グループ DFH$EXBS および DFH$EXWS に含まれるリソースを修正およびインストールすることができます。

Web クライアントの構成

Web クライアントを使用する前に、サポートされるいずれかの環境に Web クライアント・フロントエンド Java Web アーカイブ (WAR) を配置して、ご使用の CICS システムで適切なエンドポイントを呼び出すように構成しておく必要があります。

このタスクについて

次の環境は、Web クライアント・フロントエンド・アプリケーションの 2 つのバージョン、ExampleAppClientV855.war および ExampleAppWrapperClientV855.war でサポートされています。

- 最新の WebSphere Liberty Profile が含まれる CICS Liberty JVM サーバー

WAR ファイルは、z/OS UNIX の *hlq*/samples/webservices/client ディレクトリー内にあります。

手順

1. Web クライアントを始動するには、Web ブラウザーで以下の URL を入力します。その際、*mylibertyserver* は Web クライアントのインストール先 Liberty JVM サーバーのホスト名です。
   - ExampleAppClientV855.war の場合、URL http://mylibertyserver:9080/ExampleAppClientV855/ を使用します
   - ExampleAppWrapperClientV855.war の場合、URL http://mylibertyserver:9080/ExampleAppWrapperClientV855/ を使用します
2. 構成ページを表示するには、「構成 **(CONFIGURE)**」をクリックします。 構成ページが表示されます。
3. Inquire catalog、Inquire item、および Place order の 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 8080 を、TCPIPSERVICE 定義リソースで構成したポート番号に置き換えます。
4. 「**SUBMIT (**発注登録**)**」をクリックします。

タスクの結果

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。

次のタスク

Web サービス呼び出しの URL は HTTP セッションに保管されます。 したがって、Web 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。

Web サービス対応アプリケーションの実行

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。

このタスクについて

続行する前に、Web クライアントが構成されていることを確認してください。サンプル・アプリケーションの構成を参照してください。

手順

1. Web ブラウザーで、次の URL を入力します。http://*mylibertyserver*:9080/ExampleAppClientV855/。ここで、*mylibertyserver* 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。
2. 「**INQUIRE (**問い合わせ**)**」ボタンをクリックします。
3. 品目番号を入力し、「**SUBMIT (**発注登録**)**」 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始まる、カタログ内の品目のリストを表示します。

4. 注文する品目を選択します。
   a. 注文する品目の「**Select (**選択**)**」列のラジオ・ボタンをクリックします。
   b. 「**SUBMIT (**発注登録**)**」ボタンをクリックします。
5. 発注するには、以下の情報を入力します。
   a. 「**Quantity (**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. 「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. 「**Department Name (**部門名**)**」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。
   d. 「**SUBMIT (**発注登録**)**」ボタンをクリックします。

   アプリケーションにより、注文が行われたことが確認されます。

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、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。

このタスクについて

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、コピーブックは、Web サービス・アシスタントで使用できるようになりました。

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。

手順

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/webservices/JCL/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=mycicsserver:myport/exampleApp/inquireSingle
WSBIND=/u/exampleapp/wsbind/inquireSingle.wsbind
WSDL=/u/exampleapp/wsdl/inquireSingle.wsdl
*/
```

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 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 にインストールされているデータ・セット SDFHSAMP です。

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開始されるターゲット・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=*mycicsserver*:*myport*/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 リソースがインストールされている CICS サーバーの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 z/OS UNIX 上の場所。

注: このファイルがパイプライン・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書き込み先となる z/OS UNIX 上の場所。 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。

従来、Web サービス記述が格納されているファイルの拡張子は `.wsdl` です。

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使用する必要がある情報を提供します。 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。

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。

生成される WSDL 文書の例

Web サービス・アシスタント・プログラム DFHLS2WS の実行時に生成される Web サービス記述 (WSDL) 文書の例。

```
<?xml version="1.0" ?>
<definitions targetNamespace="http://www.DFH0XCMN.DFH0XCP4.com" xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:reqns="http://www.DFH0XCMN.DFH0XCP4.Request.com" xmlns:resns="http://www.DFH0XCMN.DFH0XCP4.Response.com"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:tns="http://www.DFH0XCMN.DFH0XCP4.com">
    <types>
        <xsd:schema attributeFormDefault="qualified" elementFormDefault="qualified"
        targetNamespace="http://www.DFH0XCMN.DFH0XCP4.Request.com" xmlns:tns="http://www.DFH0XCMN.DFH0XCP4.Request.com"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema">
            <xsd:complextype abstract="false" block="#all" final="#all" mixed="false" name="ProgramInterface">
                <xsd:annotation>
                    <xsd:documentation source="http://www.ibm.com/software/htp/cics/annotations">
                    This schema was generated by the CICS web services assistant.
                    </xsd:documentation>
                </xsd:annotation>
                <xsd:sequence>
                    <xsd:element name="ca_request_id" nillable="false">
                        <xsd:simpletype>
                            <xsd:annotation>
                                <xsd:appinfo source="http://www.ibm.com/software/htp/cics/annotations">
                                #Thu Nov 03 11:55:26 GMT 2005 com.ibm.cics.wsdl.properties.synchronized=false
                                </xsd:appinfo>
                            </xsd:annotation>
                            <xsd:restriction base="xsd:string">
```

```
                                <xsd:maxlength value="6"/>
                                <xsd:whitespace value="preserve"/>
                            </xsd:restriction>
                        </xsd:simpletype>
                    </xsd:element>

.... most of the schema for the request is removed

                </xsd:sequence>
            </xsd:complextype>
            <xsd:element name="DFH0XCMNOperation" nillable="false" type="tns:ProgramInterface"/>
        </xsd:schema>
        <xsd:schema attributeFormDefault="qualified" elementFormDefault="qualified"
        targetNamespace="http://www.DFH0XCMN.DFH0XCP4.Response.com" xmlns:tns="http://www.DFH0XCMN.DFH0XCP4.Response.com"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema">

... schema content for the reply is removed

        </xsd:schema>
    </types>
    <message name="DFH0XCMNOperationResponse">
        <part element="resns:DFH0XCMNOperationResponse" name="ResponsePart"/>
    </message>
    <message name="DFH0XCMNOperationRequest">
        <part element="reqns:DFH0XCMNOperation" name="RequestPart"/>
    </message>
    <porttype name="DFH0XCMNPort">
        <operation name="DFH0XCMNOperation">
            <input message="tns:DFH0XCMNOperationRequest" name="DFH0XCMNOperationRequest"/>
            <output message="tns:DFH0XCMNOperationResponse" name="DFH0XCMNOperationResponse"/>
        </operation>
    </porttype>
    <binding name="DFH0XCMNHTTPSoapBinding" type="tns:DFH0XCMNPort">
        <!-- This soap:binding indicates the use of SOAP 1.1 -->
        <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
        <!-- This soap:binding indicates the use of SOAP 1.2 -->
        <!-- <soap:binding style="document" transport="http://www.w3.org/2003/05/soap-http"/> -->
        <operation name="DFH0XCMNOperation">
            <soap:operation soapAction="" style="document"/>
            <input name="DFH0XCMNOperationRequest">
            <soap:body parts="RequestPart" use="literal"/>
            </input>
            <output name="DFH0XCMNOperationResponse">
                <soap:body parts="ResponsePart" use="literal"/>
            </output>
        </operation>
    </binding>
    <service name="DFH0XCMNService">
        <port binding="tns:DFH0XCMNHTTPSoapBinding" name="DFH0XCMNPort">
            <!-- This soap:address indicates the location of the web service over HTTP.
                 Please replace "my-server" with the TCPIP host name of your CICS region.
                 Please replace "my-port" with the port number of your CICS TCPIPSERVICE. -->
            <soap:address location="http://my-server:my-port/exampleApp/inquireSingles.log"/>
            <!-- This soap:address indicates the location of the web service over HTTPS. -->
            <!-- <soap:address location="https://my-server:my-port/exampleApp/inquireSingles.log"/> -->
            <!-- This soap:address indicates the location of the web service over Websphere MQSeries.
                 Please replace "my-queue" with the appropriate queue name. -->
            <!-- <soap:address location="jms:/queue?destination=my-queue&amp;connectionFactory=()&amp;
            targetService=/exampleApp/inquireSingles.log&amp;initialContextFactory=com.ibm.mq.jms.Nojndi" /> -->
        </port>
    </service>
</definitions>
```

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 WEBSERVICE 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。

このタスクについて

パイプライン・スキャン・コマンドを実行すると、CICS はピックアップ・ディレクトリーをスキャンして、.wsbind という拡張子を持つ WEBSERVICE バインディング・ファイル検索します。 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 リソースと、Web サービスのインストール先 PIPELINE に URI がマップされます。 スキャンされた WEBSERVICE リソースが破棄されると、これに関連付けられている URIMAP リソースも破棄されます。

手順

1. 「定義 **(Definitions)**」 > 「パイプライン定義 **(Pipeline Definitions)**」を選択して、CICS Explorer でプロバイダー・パイプライン PIPELINE(EXPIPE01) の PIPELINE 定義を変更します。 EXPIPE01 をダブルクリックし、「パイプライン定義 (EXPIPE01) (Pipeline Definition (EXPIPE01))」エディターを開きます。「属性 **(Attributes)**」タブで、「**WS ディレクトリー (WS Directory)**」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 WEBSERVICE 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 WEBSERVICE 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、次のファイルがコピーされます。

   ```
   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 リソースをインストールします。

タスクの結果

CICS は 2 つの URIMAP リソースを作成します。1 つ目の URIMAP 定義は、要求を処理する他のリソース (PIPELINE リソースなど) にインバウンド Web サービス要求の URI をマップする情報がこの定義に含まれる場合に、サービス・プロバイダーで必要になります。 2 つ目の URIMAP には、Web サービスに関連付けられた WSDL 文書のインバウンド要求の URI をマップする情報が含まれています。

ベース・アプリケーションのコンポーネント

以下の表を使用して、SDFHSAMP サンプルで提供されているベース・アプリケーションおよびメンバーのコンポーネントについて理解します。 リストされている SDFHSAMP メンバーには、ベース・アプリケーション、Web サービス・クライアント・アプリケーション、およびラッパー・モジュールの BMS マップ、COBOL ソース、およびコピーブックが含まれています。

表 *18. BMS* マップを含む *SDFHSAMP* メンバー

メンバー名	説明
DFH0XS1	「**Main Menu (**メインメニュー**)**」画面のマップ (EXMENU) および「**Details of your order (**注文の詳細**)**」画面のマップ (EXORDR) で構成されるマップ・セットの BMS マクロ。
DFH0XS2	「**Inquire Catalog (**カタログの問い合わせ**)**」画面のマップ (EXINQC) で構成されるマップ・セットの BMS マクロ。

表 *18. BMS* マップを含む *SDFHSAMP* メンバー *(*続き*)*

メンバー名	説明
DFH0XS3	「**Configure CICS example catalog application (CICS** 実例カタログ・アプリケーションの構成**)**」画面のマップ (EXCONF) で構成されるマップ・セットの BMS マクロ。
DFH0XM1	DFH0XS1 をアセンブルすることによって生成される COBOL コピーブック。 DFH0XGUI および DFH0XCUI には、このコピーブックが組み込まれています。
DFH0XM2U	DFH0XS2 をアセンブルして、コピーブックのプログラミングを容易にするために索引付きの配列構造を組み込むようその結果を編集することによって生成される COBOL コピーブック。 DFH0XGUI および DFH0XCUI には、このコピーブックが組み込まれています。
DFH0XM3	DFH0XS3 をアセンブルすることによって生成される COBOL コピーブック。 DFH0XCFG には、このコピーブックが組み込まれています。

表 *19.* ベース・アプリケーションの *COBOL* ソースが含まれる *SDFHSAMP* メンバー

メンバー名	説明
DFH0XCFG	VSAM 構成ファイルを読み取って更新するために、トランザクション ECFG によって呼び出されるプログラム。
DFH0XCMN	カタログ・アプリケーションのコントローラー・プログラム。すべての要求がコントローラー・プログラムをパススルーします。
DFH0XGUI	端末ユーザーへの BMS マップの送信および端末ユーザーからのマップの受信を管理するために、トランザクション EGUI によって呼び出されるプログラム。 このプログラムはプログラム DFH0XCMN にリンクします。
DFH0XODE	注文発送 Web サービスの 2 つのバージョンのエンドポイントのいずれか。 これは、CICS で稼働するバージョンです。このプログラムは、戻りの COMMAREA でテキスト「`Order in dispatch`」を設定します。
DFH0XSDS	VSAM カタログ・ファイルがセットアップされていない場合に、アプリケーションが操作できるスタブ化 版またはダミー版のデータ・ストア・プログラム。 DFH0XSDS は、VSAM ファイルに保管されたデータではなく、プログラムで定義されたデータを使用します。
DFH0XSOD	スタブ化版の注文発送プログラム。このプログラムは、COMMAREA 内の戻りコードを 0 に設定して、呼び出し元に戻します。 DFH0XSOD は、アウトバウンド Web サービスが必要ないときに使用されます。
DFH0XSSM	スタブ化版の在庫マネージャー (補充) プログラム。 DFH0XSSM は、COMMAREA 内の戻りコードを 0 に設定して、呼び出し元に戻します。
DFH0XVDS	VSAM 版のデータ・ストア・プログラム。 DFH0XVDS は VSAM ファイルにアクセスして、カタログの読み取りと更新を実行します。

表 19. ベース・アプリケーションの *COBOL* ソースが含まれる *SDFHSAMP* メンバー (続き)

メンバー名	説明
DFH0XWOD	Web サービス版の注文発送プログラム。 DFH0XWOD は EXEC CICS INVOKE WEBSERVICE を発行して、注文発送プログラムへのアウトバウンド Web サービス呼び出しを行います。

表 20. ベース・アプリケーションの *COBOL* コピーブックが含まれる *SDFHSAMP* メンバー

メンバー名	説明
DFH0XCP1	Inquire Catalog、Inquire Single、および Place Order の機能の要求と応答が含まれる COMMAREA 構造を定義します。 プログラム DFH0XCMN、DFH0XCUI、DFH0XECC、DFH0XGUI、DFH0XICW、DFH0XISW、DFH0XPOW、DFH0XSDS、および DFH0XVDS には、このコピーブックが組み込まれています。
DFH0XCP2	注文発送モジュールと在庫マネージャー・モジュールの COMMAREA 構造を定義します。 プログラム DFH0XCMN、DFH0XSOD、DFH0XSSM、および DFH0XWOD には、このコピーブックが組み込まれています。
DFH0XCP3	Inquire Catalog の要求と応答のデータ構造を定義します。 `inquireCatalog.wsdl` および `inquireCatalog.wsbind` を生成するために、DFHLS2WS への入力として使用されます。
DFH0XCP4	Inquire Single の要求と応答のデータ構造を定義します。 `inquireSingle.wsdl` および `inquireSingle.wsbind` を生成するために、DFHLS2WS への入力として使用されます。
DFH0XCP5	Place Order の要求と応答のデータ構造を定義します。 `placeOrder.wsdl` および `placeOrder.wsbind` を生成するために、DFHLS2WS への入力として使用されます。
DFH0XCP6	Dispatch Order の要求と応答のデータ構造を定義します。 `dispatchOrder.wsdl` および `dispatchOrder.wsbind` を生成するために、DFHLS2WS への入力として使用されます。
DFH0XCP7	Dispatch Order 要求のデータ構造を定義します。 プログラム DFH0XODE および DFH0XWOD には、このコピーブックが組み込まれています。
DFH0XCP8	Dispatch Order 応答のデータ構造を定義します。 プログラム DFH0XODE および DFH0XWOD には、このコピーブックが組み込まれています。

表 21. *CICS* で稼働する *Web* サービス・クライアント・アプリケーションの *COBOL* ソース・コードを含む *SDFHSAMP* メンバー

メンバー名	説明
DFH0XCUI	端末ユーザーへの BMS マップの送信および端末ユーザーからのマップの受信を管理するために、トランザクション ECLI によって起動されるプログラム。 これはプログラム DFH0XECC にリンクします。

表 21. *CICS* で稼働する *Web* サービス・クライアント・アプリケーションの *COBOL* ソース・コードを含む *SDFHSAMP* メンバー (続き)

メンバー名	説明

DFH0XECC	EXEC CICS INVOKE WEBSERVICE コマンドを使用して、ベース・アプリケーションにアウトバウンド Web サービス要求を行います。 指定される Web サービスは、以下のいずれかです。
	`inquireCatalogClient`
	`inquireSingleClient`
	`placeOrderClient`

表 22. *CICS* で稼働する *Web* サービス・クライアント・アプリケーションの *COBOL* コピーブックを含む *SDFHSAMP* メンバー： これらのメンバーはすべて DFHWS2LS によって生成され、プログラム DFH0XECC によって組み込まれます。

メンバー名	説明
DFH0XCPA	Inquire Catalog 要求のデータ構造を定義します。
DFH0XCPB	Inquire Catalog 応答のデータ構造を定義します。
DFH0XCPC	Inquire Single 要求のデータ構造を定義します。
DFH0XCPD	Inquire Single 応答のデータ構造を定義します。
DFH0XCPE	Place Order 要求のデータ構造を定義します。
DFH0XCPF	Place Order 応答のデータ構造を定義します。

表 23. ラッパー・モジュールの *COBOL* ソース・コードが含まれる *SDFHSAMP* メンバー

メンバー名	説明
DFH0XECC	Web サービス・クライアント・プログラム
DFH0XICW	`inquireCatalog` サービスのラッパー・プログラム。
DFH0XISW	`inquireSingle` サービスのラッパー・プログラム。
DFH0XPOW	`purchaseOrder` サービスのラッパー・プログラム。

表 24. ラッパー・モジュールの *COBOL* コピーブックが含まれる *SDFHSAMP* メンバー

メンバー名	説明
DFH0XWC1	Inquire Catalog 要求のデータ構造を定義します。 プログラム DFH0XICW には、このコピーブックが組み込まれています。
DFH0XWC2	Inquire Catalog 応答のデータ構造を定義します。 プログラム DFH0XICW には、このコピーブックが組み込まれています。
DFH0XWC3	Inquire Single 要求のデータ構造を定義します。 プログラム DFH0XISW には、このコピーブックが組み込まれています。
DFH0XWC4	Inquire Single 応答のデータ構造を定義します。 プログラム DFH0XISW には、このコピーブックが組み込まれています。
DFH0XWC5	Place Order 要求のデータ構造を定義します。 プログラム DFH0XPOW には、このコピーブックが組み込まれています。
DFH0XWC6	Place Order 応答のデータ構造を定義します。 プログラム DFH0XPOW には、このコピーブックが組み込まれています。

表 25. CICS リソース定義

リソース名	リソース・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、アプリケーションに対する 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)
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・カタログの EXMPCAT VSAM ファイルのファイル定義 (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のファイル定義。

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ようにするため、カタログ・マネージャーが実行する型検査とエラー・リカバリーは制限されています。

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:

標準の CICS 通信域 (COMMAREA) を使用して、サンプルのクライアント・プログラムとサーバー・プログラムの間でデータが渡されます。

以下のコードの抜粋では、カタログ・マネージャー・アプリケーションの COMMAREA 構造が示されています。

```
*    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).
          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。

表 *26.* カタログ・マネージャーの戻りコード

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。

表 26. カタログ・マネージャーの戻りコード (続き)

タイプ	コード	説明
カタログ・ファイル	20	品目の参照番号が見つかりませんでした。
	21	カタログ・ファイルの参照のオープン、読み取り、または終了時のエラーです。
	22	ファイルの更新エラーです。
構成ファイル	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・タイプが STUB と VSAM のいずれでもありませんでした。
	52	アウトバウンド Web サービスの切り替えが Y と N のいずれでもありませんでした。
リモート Web サービス	30	**EXEC CICS INVOKE WEBSERVICE** コマンドが INVREQ 状態を戻しました。
	31	**EXEC CICS INVOKE WEBSERVICE** コマンドが NOTFND 状態を戻しました。
	32	**EXEC CICS INVOKE WEBSERVICE** コマンドが INVREQ または NOTFND 以外の状態を戻しました。
アプリケーション	97	注文を完了するには在庫が不足しています。
	98	注文数量が正数ではありませんでした。
	99	DFH0XCMN が、CA-REQUEST-ID フィールドが 01INQC、01INQS、または 01ORDR のいずれかに設定されていない COMMAREA を受け取りました。

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
操作を指定するストリング。

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
操作を指定するストリング。

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-SINGLE-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
操作を指定するストリング。

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-ORD-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-ORD-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-ORD-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-ORD-ITEM-REF-NUMBER
注文された品目の参照番号。

CA-ORD-QUANTITY-REQ
品目の必要数。

出力パラメーター

CA-ORD-RETURN-CODE
操作を指定するストリング。

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-ORD-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-STK-ITEM-REF-NUMBER
注文された品目の参照番号。

CA-STK-QUANTITY-REQ
品目の必要数。

出力パラメーター

CA-ORD-RETURN-CODE
操作を指定するストリング。

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。

カタログ・ファイル・レコード

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・レコード

構成ファイルは、4 つの異なるレコードを持つ KSDS VSAM ファイルです。

表 27. 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

表 27. 一般情報レコード (続き)

名前	**COBOL** データ・タイプ	説明
充てん文字	PIC X(10)	

表 28. 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 29. カタログ・ファイル情報レコード

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

表 30. サーバー情報レコード

名前	**COBOL** データ・タイプ	説明
WS-SERVER-KEY	PIC X(9)	WS-SERVER を含むサーバー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)	CICS Web サービス・クライアントの場合に限り、実例アプリケーションが Web サービスとして配置されているシステムの IP アドレスとポート

JSON のサンプル

これらの例は、JSON 要求について理解するうえで役立ちます。

照会ストリングを使用した HTTP GET 要求の例

以下は、照会ストリング使用した HTTP GET 要求の例です。

```
GET /genapp/customers?name=Joe%20Bloggs/ 1
Host: www.example.com
```

ここで、 **1** は照会ストリングです。

JSON 本体を使用した HTTP 要求例

以下は、JSON 本体を使用した HTTP 要求の例です。

```
 POST /genapp/customers/
Host: www.example.com
Content-Type: application/json
Content-Length: nn 1

{
   "customers":
  {
    "firstName": "Joe",
    "lastName": "Bloggs",
    "fullAddress":
    {
        "streetAddress": "21 2nd Street",
        "city": "New York",
        "state": "NY",
        "postalCode": 10021
    }
  }
}
```

ここで、Content-Length:*nn* 1 は要求の長さです。

この例の COBOL 言語構造マッピングは以下のとおりです。

```
01 CUSTOMERS.
    03 firstname pic x(8).
    03 lastname pic x(8).
    03 fulladdress.
      05 streetaddress pic x(20).
      05 city pic x(20).
      05 state pic xx.
      05 postalcode pic 9(5).
```

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。この資料の他の言語版を IBM から入手できる場合があります。ただし、これを入手するには、本製品または当該言語版製品を所有している必要がある場合があります。

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。 IBM 製品、プログラムまたはサービスに代えて、IBM の知的所有権を侵害することのない機能的に同等のプログラムまたは製品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。

〒*103-8510*
東京都中央区日本橋箱崎町*19*番*21*号
日本アイ・ビー・エム株式会社
法務・知的財産
知的財産権ライセンス渉外

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。

この情報には、技術的に不適切な記述や誤植を含む場合があります。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。

本書において IBM 以外の Web サイトに言及している場合がありますが、便宜のため記載しただけであり、決してそれらの Web サイトを推奨するものではありません。それらの Web サイトにある資料は、この IBM 製品の資料の一部ではありません。それらの Web サイトは、お客様自身の責任でご使用ください。

IBM は、お客様が提供するいかなる情報も、お客様に対してなんら義務も負うことのない、自ら適切と信ずる方法で、使用もしくは配布することができるものとします。

© Copyright IBM Corp. 2005, 2018

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 Director of Licensing
IBM Corporation
North Castle Drive, MD-NC119 Armonk,
NY 10504-1785
United States of America

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。

IBM 以外の製品に関する情報は、その製品の供給者、出版物、もしくはその他の公に利用可能なソースから入手したものです。IBM は、それらの製品のテストは行っておりません。したがって、他社製品に関する実行性、互換性、またはその他の要求については確証できません。 IBM 以外の製品の性能に関する質問は、それらの製品の供給者にお願いします。

本書には、日常の業務処理で用いられるデータや報告書の例が含まれています。より具体性を与えるために、それらの例には、個人、企業、ブランド、あるいは製品などの名前が含まれている場合があります。これらの名前はすべて架空のものであり、類似する個人や企業が実在しているとしても、それは偶然にすぎません。

著作権使用許諾:

本書には、様々なオペレーティング・プラットフォームでのプログラミング手法を例示するサンプル・アプリケーション・プログラムがソース言語で掲載されています。お客様は、サンプル・プログラムが書かれているオペレーティング・プラットフォームのアプリケーション・プログラミング・インターフェースに準拠したアプリケーション・プログラムの開発、使用、販売、配布を目的として、いかなる形式においても、IBM に対価を支払うことなくこれを複製し、改変し、配布することができます。このサンプル・プログラムは、あらゆる条件下における完全なテストを経ていません。従って IBM は、これらのサンプル・プログラムについて信頼性、利便性もしくは機能性があることをほのめかしたり、保証することはできません。これらのサンプル・プログラムは特定物として現存するままの状態で提供されるものであり、いかなる保証も提供されません。 IBM は、お客様の当該サンプル・プログラムの使用から生ずるいかなる損害に対しても一切の責任を負いません。

プログラミング・インターフェース情報

CICS には、プログラミング・インターフェースと見なすことのできる資料と、プログラミング・インターフェースと見なすことのできない資料があります。

オンライン製品資料の以下のセクションには、CICS Transaction Server for z/OS, バージョン 5 リリース 5 のサービスを取得するプログラムをお客様が作成するためのプログラミング・インターフェースが含まれています。

- アプリケーションの開発
- Developing system programs
- 保護の概要
- 外部インターフェースに向けた開発
- リファレンス: アプリケーション開発h
- リファレンス: システム・プログラミング
- リファレンス: 接続

オンライン製品資料の以下のセクションには、CICS Transaction Server for z/OS, バージョン 5 リリース 5 のプログラミング・インターフェースとして意図されていない (プログラミング・インターフェースと誤解される可能性のある) 情報が含まれています。

- Troubleshooting and support
- リファレンス: 診断

PDF 形式のマニュアルで CICS 資料にアクセスする場合は、CICS Transaction Server for z/OS, バージョン 5 リリース 5 のサービスを取得するプログラムをお客様が作成するためのプログラミング・インターフェースが以下のマニュアルに含まれています。

- アプリケーション・プログラミング・ガイドおよびアプリケーション・プログラミング・リファレンス
- Business Transaction Services
- Customization Guide
- C++ OO Class Libraries
- Debugging Tools Interfaces Reference
- Distributed Transaction Programming Guide
- External Interfaces Guide
- Front End Programming Interface Guide
- IMS Database Control Guide
- インストール・ガイド
- セキュリティー・ガイド
- Supplied Transactions
- CICSPlex SM Managing Workloads
- CICSPlex SM Managing Resource Usage
- CICSPlex SM アプリケーション・プログラミング・ガイドおよび CICSPlex SM アプリケーション・プログラミング・リファレンス
- Java Applications in CICS

PDF 形式のマニュアルで CICS 資料にアクセスする場合は、CICS Transaction Server for z/OS, バージョン 5 リリース 5 のプログラミング・インターフェース

として意図されていない (プログラミング・インターフェースと誤解される可能性のある) 情報が以下のマニュアルに含まれています。

- Data Areas
- Diagnosis Reference
- Problem Determination Guide
- CICSPlex SM Problem Determination Guide

商標

IBM、IBM ロゴおよび ibm.com® は、世界の多くの国で登録された International Business Machines Corporation の商標または登録商標です。他の製品名およびサービス名等は、それぞれ IBM または各社の商標である場合があります。現時点での IBM の商標リストについては、 http://www.ibm.com/legal/copytrade.shtml をご覧ください。

Adobe、Adobe ロゴ、PostScript、PostScript ロゴは、Adobe Systems Incorporated の米国およびその他の国における登録商標または商標です。

インテル、Intel、Intel ロゴ、Intel Inside、Intel Inside ロゴ、Intel Centrino、Intel Centrino ロゴ、Celeron、Intel Xeon、Intel SpeedStep、Itanium、および Pentium は、Intel Corporation または子会社の米国およびその他の国における商標または登録商標です。

Java およびすべての Java 関連の商標およびロゴは Oracle やその関連会社の米国およびその他の国における商標または登録商標です。

Linux は、Linus Torvalds の米国およびその他の国における登録商標です。

Microsoft、Windows、Windows NT および Windows ロゴは、Microsoft Corporation の米国およびその他の国における商標です。

UNIX は The Open Group の米国およびその他の国における登録商標です。

製品資料に関するご使用条件

これらの資料は、以下のご使用条件に同意していただける場合に限りご使用いただけます。

適用範囲
: IBM Web サイトの「ご利用条件」に加えて、以下のご使用条件が適用されます。

個人使用
: これらの資料は、すべての著作権表示その他の所有権表示をしていただくことを条件に、非商業的な個人による使用目的に限り複製することができます。ただし、IBM の明示的な承諾をえずに、これらの資料またはその一部について、二次的著作物を作成したり、配布 (頒布、送信を含む) または表示 (上映を含む) することはできません。

商用使用
: これらの資料は、すべての著作権表示その他の所有権表示をしていただくことを条件に、お客様の企業内に限り、複製、配布、および表示することがで

きます。ただし、IBM の明示的な承諾をえずにこれらの資料の二次的著作物を作成したり、お客様の企業外で資料またはその一部を複製、配布、または表示することはできません。

権利　ここで明示的に許可されているもの以外に、資料や資料内に含まれる情報、データ、ソフトウェア、またはその他の知的所有権に対するいかなる許可、ライセンス、または権利を明示的にも黙示的にも付与するものではありません。

資料の使用が IBM の利益を損なうと判断された場合や、上記の条件が適切に守られていないと判断された場合、IBM はいつでも自らの判断により、ここで与えた許可を撤回できるものとさせていただきます。

お客様がこの情報をダウンロード、輸出、または再輸出する際には、米国のすべての輸出入関連法規を含む、すべての関連法規を遵守するものとします。

IBM これらの資料の内容についていかなる保証もしません。これらの資料は、特定物として現存するままの状態で提供され、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なしで提供されます。

IBM オンラインでのプライバシー・ステートメント

サービス・ソリューションとしてのソフトウェアも含めた IBM ソフトウェア製品 (「ソフトウェア・オファリング」) では、製品の使用に関する情報の収集、エンド・ユーザーの使用感の向上、エンド・ユーザーとの対話またはその他の目的のために、Cookie はじめさまざまなテクノロジーを使用することがあります。多くの場合、ソフトウェア・オファリングにより個人情報が収集されることはありません。IBM の「ソフトウェア・オファリング」の一部には、個人情報を収集できる機能を持つものがあります。ご使用の「ソフトウェア・オファリング」が、これらの Cookie およびそれに類するテクノロジーを通じてお客様による個人情報の収集を可能にする場合、以下の具体的事項をご確認ください。

CICSPlex SM Web ユーザー・インターフェース **(**メイン・インターフェース**)** の場合**:**　このソフトウェア・オファリングは、展開される構成に応じて、セッション管理、認証、お客様の利便性の向上、または利用の追跡または機能上の目的のために、それぞれのお客様のユーザー名、およびその他の個人情報を、セッションごとの Cookie および持続的な Cookie を使用して収集する場合があります。これらの Cookie を無効にすることはできません。

CICSPlex SM Web ユーザー・インターフェース **(**データ・インターフェース**)** の場合**:**　このソフトウェア・オファリングは、展開される構成に応じて、セッション管理、認証、または利用の追跡または機能上の目的のために、それぞれのお客様のユーザー名またはその他の個人情報を、セッションごとの Cookie を使用して収集する場合があります。これらの Cookie を無効にすることはできません。

CICSPlex SM Web ユーザー・インターフェース **(**「**Hello World**」ページ**)** の場合**:**　このソフトウェア・オファリングは、展開される構成に応じて、個人情報を収集しないセッションごとの Cookie を使用する場合があります。これらの Cookie を無効にすることはできません。

CICS Explorer の場合**:**

このソフトウェア・オファリングは、展開される構成に応じて、セッション管理、お客様の利便性の向上、または利用の追跡または機能上の目的のために、それぞれのお客様のユーザー名、およびその他の個人情報を、セッションごとの設定および持続的な設定を使用して収集する場合があります。これらの設定を無効にすることはできませんが、ユーザー・パスワードの暗号化形式でのディスクへの保管は、サインオン中にチェック・ボックスにチェック・マークを付けることによるユーザーの明示的な操作によってのみ有効化することができます。

この「ソフトウェア・オファリング」が Cookie およびさまざまなテクノロジーを使用してエンド・ユーザーから個人を特定できる情報を収集する機能を提供する場合、 お客様は、このような情報を収集するにあたって適用される法律、ガイドライン等を遵守する必要があります。これには、エンドユーザーへの通知や同意の要求も含まれますがそれらには限られません。

このような目的での Cookie を含む様々なテクノロジーの使用の詳細については、『IBM オンラインでのプライバシー・ステートメント』(http://www.ibm.com/privacy/details/jp/ja/) の『クッキー、ウェブ・ビーコン、その他のテクノロジー』および「IBM Software Products and Software-as-a-Service Privacy Statement」(http://www.ibm.com/software/info/product-privacy) を参照してください。

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]

アシスタント、JSON 268, 453
アシスタント、Web サービス 535
アトミック・トランザクション 221, 229
  サービス・プロバイダーの構成 226
  サービス・リクエスターの構成 227
  状態 230
  登録サービス 222
  CICS の構成 224
アドレッシング (addressing)
  パイプライン構成エレメント 119
アプリケーション
  XML へ変換 332
アプリケーション・データの JSON への変換 528
アプリケーション・データへの JSON の変換 530
アプリケーション・データへの XML スキーマのマッピング 447
アプリケーション・ハンドラー
  パイプライン構成エレメント 114
アルゴリズム 663, 665
インストール 692
永続メッセージ 58
永続メッセージのサポート 59
エラー 697
エラー・コンテナー
  DFHJSON-ERROR 700
  DFHJSON-ERRORMSG 701
エンドポイント参照
  デフォルト 251
エンドポイント参照 (EPR) 243
エンベロープ、SOAP 22

[カ行]

カスタムのセキュリティー・ハンドラー 675
カタログ・サンプルカタログ・サンプル 703
起動、Trust クライアント 677
機能 46
繰り返しコンテンツ 403, 433
グローバル・ユーザー出口 217
言語構造からのマッピングの生成 524
構成 84, 90
構成、パイプラインの 672
構成ファイル、パイプライン 101
互換モード 233
コマンド行 697
コンテキスト・コンテナー 188
  DFHWS-CID-DOMAIN 198
  DFHWS-IDTOKEN 204
  DFHWS-LOCATION 191
  DFHWS-MEP 191
  DFHWS-MTOM-IN 199
  DFHWS-MTOM-OUT 199
  DFHWS-RESPWAIT 192
  DFHWS-RESTOKEN 204
  DFHWS-SERVICEURI 205
  DFHWS-STSACTION 205
  DFHWS-STSFAULT 205
  DFHWS-STSURI 206
  DFHWS-TOKENTYPE 206
  DFHWS-WSDL-CTX 201
  DFHWS-XOP-IN 201
  DFHWS-XOP-OUT 201
  DFH-EXIT-HEADER1 188
コンテナー
  コンテキスト・コンテナー
    DFHWS-APPHANDLER 189, 190, 192
    DFHWS-DATA 190
    DFHWS-FAULT 190
    DFHWS-PIPELINE 192
    DFHWS-SOAPLEVEL 193
    DFHWS-STSREASON 205
    DFHWS-TRANID 193
    DFHWS-URI 193
    DFHWS-USERID 198
    DFHWS-WEBSERVICE 198
    DFH-HANDLERPLIST 189
    DFH-SERVICEPLIST 189
  制御コンテナー
    DFHERROR 178
    DFHFUNCTION 180
    DFHHTTPSTATUS 183
    DFHMEDIATYPE 183
    DFHNORESPONSE 183
    DFHREQUEST 184
    DFHRESPONSE 184
    DFHWS-CCSID 185
    DFHWS-NODEJSAPP 185
  チャネル記述 317, 631
  DFHERROR 178
コンテナー (続き)
  DFHFUNCTION 180
  DFHHTTPSTATUS 183
  DFHJSON-DATA 700
  DFHJSON-JSON 699
  DFHJSON-JVMSERVR 700
  DFHJSON-TRANSFRM 700
  DFHMEDIATYPE 183
  DFHNORESPONSE 183
  DFHREQUEST 184
  DFHRESPONSE 184
  DFHSAML-AnnnVmmm 206
  DFHSAML-ASSQNAME 206
  DFHSAML-ATTRAnnn 207
  DFHSAML-ATTRFnnn 207
  DFHSAML-ATTRNnnn 207
  DFHSAML-ATTRSnnn 207
  DFHSAML-ATTRYnnn 207
  DFHSAML-AUDNRnnn 207
  DFHSAML-AUTHMETH 208
  DFHSAML-CERTIDN 208
  DFHSAML-CERTSDN 208
  DFHSAML-CERTSNUM 208
  DFHSAML-CONFMETH 208
  DFHSAML-COUNTS 208
  DFHSAML-FLAGS 208
  DFHSAML-ISSUER 208
  DFHSAML-NAMID 208
  DFHSAML-NAMIDF 209
  DFHSAML-NAMIDQ 209
  DFHSAML-NAMIDSP 209
  DFHSAML-NAMIDSPQ 209
  DFHSAML-OUTTOKEN 209
  DFHSAML-PROXYnnn 209
  DFHSAML-RESPONSE 209
  DFHSAML-SAMLID 209
  DFHSAML-SUBJADDR 209
  DFHSAML-SUBJDNS 210
  DFHSAML-TIMES 210
  DFHWS-APPHANDLER 189, 190, 192
  DFHWS-CCSID 185
  DFHWS-CID-DOMAIN 198
  DFHWS-DATA 190
  DFHWS-FAULT 190
  DFHWS-IDTOKEN 204
  DFHWS-LOCATION 191
  DFHWS-MEP 191
  DFHWS-MTOM-IN 199
  DFHWS-MTOM-OUT 199
  DFHWS-NODEJSAPP 185

© Copyright IBM Corp. 2005, 2018

コンテナー (続き)
DFHWS-PIPELINE 192
DFHWS-RESPWAIT 192
DFHWS-RESTOKEN 204
DFHWS-SERVICEURI 205
DFHWS-SOAPLEVEL 193
DFHWS-STSACTION 205
DFHWS-STSFAULT 205
DFHWS-STSREASON 205
DFHWS-STSURI 206
DFHWS-TOKENTYPE 206
DFHWS-TRANID 193
DFHWS-URI 193
DFHWS-USERID 198
DFHWS-WEBSERVICE 198
DFHWS-XOP-IN 201
DFHWS-XOP-OUT 201
DFH-EXIT-HEADER1 188
DFH-HANDLERPLIST 189
DFH-SERVICEPLIST 189
SAML 206
コンテナー DFHJSON-DATA 700
コンテナー DFHJSON-ERROR 700
コンテナー DFHJSON-ERRORMSG 701
コンテナー DFHJSON-JSON 699
コンテナー DFHJSON-JVMSERVR 700
コンテナー DFHJSON-TRANSFRM 700
コンテナー DFHSAML-AnnnVmmm 206
コンテナー DFHSAML-ASSQNAME 206
コンテナー DFHSAML-ATTRAnnn 207
コンテナー DFHSAML-ATTRFnnn 207
コンテナー DFHSAML-ATTRNnnn 207
コンテナー DFHSAML-ATTRSnnn 207
コンテナー DFHSAML-ATTRYnnn 207
コンテナー DFHSAML-AUDNRnnn 207
コンテナー DFHSAML-AUTHMETH 208
コンテナー DFHSAML-CERTIDN 208
コンテナー DFHSAML-CERTSDN 208
コンテナー DFHSAML-CERTSNUM 208
コンテナー DFHSAML-CONFMETH 208
コンテナー DFHSAML-COUNTS 208
コンテナー DFHSAML-FLAGS 208
コンテナー DFHSAML-ISSUER 208
コンテナー DFHSAML-NAMID 208
コンテナー DFHSAML-NAMIDF 209
コンテナー DFHSAML-NAMIDQ 209
コンテナー DFHSAML-NAMIDSP 209
コンテナー DFHSAML-NAMIDSPQ 209
コンテナー DFHSAML-OUTTOKEN 209
コンテナー DFHSAML-PROXYnnn 209
コンテナー DFHSAML-RESPONSE 209
コンテナー DFHSAML-SAMLID 209
コンテナー DFHSAML-SUBJADDR 209
コンテナー DFHSAML-SUBJDNS 210
コンテナー DFHSAML-TIMES 210
コンテナー DFHWS-CID-DOMAIN 198
コンテナー DFHWS-IDTOKEN 204
コンテナー DFHWS-LOCATION 191
コンテナー DFHWS-MEP 191
コンテナー DFHWS-MTOM-IN 199
コンテナー DFHWS-MTOM-OUT 199
コンテナー DFHWS-RESPWAIT 192
コンテナー DFHWS-RESTOKEN 204
コンテナー DFHWS-SERVICEURI 205
コンテナー DFHWS-STSACTION 205
コンテナー DFHWS-STSFAULT 205
コンテナー DFHWS-STSURI 206
コンテナー DFHWS-TOKENTYPE 206
コンテナー DFHWS-WSDL-CTX 201
コンテナー DFHWS-XOP-IN 201
コンテナー DFHWS-XOP-OUT 201
コンテナー DFH-EXIT-HEADER1 188

[サ行]

サービス
パイプライン構成エレメント 134
サービス・パラメーター・リスト
<service_parameter_list> 136
サービス・プロバイダー
問題の診断 684
サービス・プロバイダー・アプリケーション 313
アトミック・トランザクションの使用 226
作成、データ構造を基にして 315, 628
サービス・リクエスター
パイプライン定義 112
問題の診断 686
サービス・リクエスター・アプリケーション
アトミック・トランザクションの使用 227
サービス・リクエスター・アプリケーションの作成 532
サンプル 703
支援機能、 XML 333
実行時の制限 214
障害、SOAP 22
スキーマ
チャネル記述 317, 631
制御コンテナー 178
制限、実行時の 214
セキュリティー・コンテナー 204
DFHSAML-AnnnVmmm 206
DFHSAML-ASSQNAME 206
DFHSAML-ATTRAnnn 207
DFHSAML-ATTRFnnn 207
DFHSAML-ATTRNnnn 207
DFHSAML-ATTRSnnn 207
DFHSAML-ATTRYnnn 207
DFHSAML-AUDNRnnn 207
セキュリティー・コンテナー (続き)
DFHSAML-AUTHMETH 208
DFHSAML-CERTIDN 208
DFHSAML-CERTSDN 208
DFHSAML-CERTSNUM 208
DFHSAML-CONFMETH 208
DFHSAML-COUNTS 208
DFHSAML-FLAGS 208
DFHSAML-ISSUER 208
DFHSAML-NAMID 208
DFHSAML-NAMIDF 209
DFHSAML-NAMIDQ 209
DFHSAML-NAMIDSP 209
DFHSAML-NAMIDSPQ 209
DFHSAML-OUTTOKEN 209
DFHSAML-PROXYnnn 209
DFHSAML-RESPONSE 209
DFHSAML-SAMLID 209
DFHSAML-SUBJADDR 209
DFHSAML-SUBJDNS 210
DFHSAML-TIMES 210
セキュリティー・ハンドラー
独自の作成 675

[タ行]

端末以外のメッセージ・ハンドラー 166, 167, 168
チャネル記述 317, 631
チャネル・インターフェース 317, 631
直接モード 233
デフォルト EPR 251
WSDL 1.1 251
動的スクリプト 692
動的ルーティング
サービス・プロバイダーの場合 176
端末ハンドラーでの 176
トラブルシューティング 692, 697

[ハ行]

バイナリー添付ファイル
パイプライン構成 138
バイナリー・データの XML への変換 449
バイナリー・データへの XML の変換 450
パイプライン構成
MTOM/XOP 138
Web Services Security 144
パイプライン構成エレメント
<addressing> 119
<apphandler> 116
<authentication> 147
<auth_token_type> 156

パイプライン構成エレメント (続き)
<cics_json_handler_java> 120
<cics_mtom_handler> 139
<cics_soap_1.1_handler> 121
<cics_soap_1.1_handler_java> 123
<cics_soap_1.2_handler> 126
<cics_soap_1.2_handler_java> 128
<default_http_transport_handler_list> 131
<default_mq_transport_handler_list> 131
<default_transport_handler_list> 132
<dfhmtom_configuration> 140
<dfhwsse_configuration> 146
<encrypt_body> 159
<handler> 132
<jvmserver> 133
<mime_options> 144
<mtom> 138
<mtom_options> 140
<named_transport_entry> 115
<namespace> 119
<provider_pipeline> 116
<provider_pipeline_json> 116
<repository> 134
<requester_pipeline> 119
<service> 134
<service_handler_list> 135
<sign_body> 158
<sts_authentication> 153
<sts_endpoint> 157
<terminal_handler> 117
<transport> 137
<transport_handler_list> 118
<wsse_handler> 145
<xop_options> 142
パイプライン構成ファイル 101
パイプライン処理
カスタマイズ 217
URI の指定変更 219
パイプライン処理のカスタマイズ 217
パイプライン定義
サービス・リクエスター 112
バインディング・ファイル
XML バインディング 445
バッチ・ユーティリティー 268
JSON アシスタント 453
Web サービス・アシスタント 535
XML 支援機能 333
ハンドラー
パイプライン構成エレメント 132
ヘッダー、SOAP 22
変数配列 324, 389, 406, 413, 419, 517, 615
本体、SOAP 22

[マ行]

メッセージ交換パターン (MEP) 14
メッセージ・ハンドラー
起動、Trust クライアント 677
端末以外の 166, 167, 168
戻りコード 697
問題の診断
サービス・プロバイダー 684
サービス・リクエスター 686
問題判別 692

[ヤ行]

ユーザー・コンテナー 211
ユーティリティー・プログラム
JSON アシスタント 268, 453
Web サービス・アシスタント 535
XML 支援機能 333

[ラ行]

リンク可能インターフェースを使用した JSON の変換 452
例 703

[ワ行]

ワークロード管理
サービス・プロバイダーの場合 176
端末ハンドラーでの 176

A

apphandler
パイプライン構成エレメント 116
Atom フィード
XML バインディング
作成 445
authentication
パイプライン構成エレメント 147
auth_token_type
パイプライン構成エレメント 156
Axis2 65

C

C および C++
JSON スキーマへのマッピング 492, 496, 590, 594
XML スキーマへのマッピング 369, 372
C および C++ へのマッピング 369, 372, 492, 496, 590, 594
CICS Web サービス
Unicode 438, 522, 622
UTF-16 438, 522, 622
cics_json_handler_java
パイプライン構成エレメント 120
cics_mtom_handler
パイプライン構成エレメント 139
cics_soap_1.1_handler
パイプライン構成エレメント 121
cics_soap_1.1_handler_java
パイプライン構成エレメント 123
cics_soap_1.2_handler
パイプライン構成エレメント 126
cics_soap_1.2_handler_java
パイプライン構成エレメント 128
COBOL
可変の繰り返しコンテンツ 403, 433
JSON スキーマへのマッピング 482, 488, 580, 586
XML スキーマへのマッピング 357, 364
COBOL へのマッピング 357, 364, 482, 488, 580, 586

D

default_http_transport_handler_list
パイプライン構成エレメント 131, 133, 134
default_mq_transport_handler_list
パイプライン構成エレメント 131
default_target
パイプライン構成エレメント 137
default_transport_handler_list
パイプライン構成エレメント 132
DFHAXIS 700
DFHERROR コンテナー 178
DFHFUNCTION コンテナー 180
DFHHTTPSTATUS コンテナー 183
DFHJS2LS
カタログ式プロシージャー 282, 297, 464
DFHLS2SC、CICS XML 支援機能プログラム 445
DFHMEDIATYPE コンテナー 183
dfhmtom_configuration
パイプライン構成エレメント 140
DFHNORESPONSE コンテナー 183
DFHREQUEST コンテナー 184
DFHRESPONSE コンテナー 184
DFHSC2LS
カタログ式プロシージャー 342
DFHWS2LS
カタログ式プロシージャー 553
dfhwsse_configuration
パイプライン構成エレメント 146

DFHWS-APPHANDLER コンテナー 189, 190, 192
DFHWS-CCSID コンテナー 185
DFHWS-DATA コンテナー 190
DFHWS-FAULT コンテナー 190
DFHWS-NODEJSAPP コンテナー 185
DFHWS-PIPELINE コンテナー 192
DFHWS-SOAPLEVEL コンテナー 193
DFHWS-STSREASON コンテナー 205
DFHWS-TRANID コンテナー 193
DFHWS-URI コンテナー 193
DFHWS-USERID コンテナー 198
DFHWS-WEBSERVICE コンテナー 198
DFH-HANDLERPLIST コンテナー 189
DFH-SERVICEPLIST コンテナー 189

E

encrypt_body
  パイプライン構成エレメント 159
EXEC CICS SOAPFAULT CREATE コマンド 635

G

GLUE 217

J

Java 65
  Web サービスの呼び出し 649
Java からの Web サービスの呼び出し 649
Java ベースの SOAP パイプライン 246
JCICS コンテナー
  DFHJSON-DATA 700
  DFHJSON-ERROR 700
  DFHJSON-ERRORMSG 701
  DFHJSON-JSON 699
  DFHJSON-JVMSERVR 700
  DFHJSON-TRANSFRM 700
  JSON 699, 700
JSON 84, 90, 197, 692, 694, 695, 696, 697, 699, 700, 701, 749, 750
  言語構造への変換 464
JSON アシスタント 268, 453, 694, 697
JSON スキーマ 313, 482, 488, 492, 496, 500, 506, 580, 586, 590, 594, 598, 604
JSON スキーマからのマッピングの生成 526
JSON のサンプル 749, 750
JVM server (JVM サーバー) 65, 649

M

maxOccurs
  JSON スキーマ内 324, 413, 517
  XML スキーマ内で 389, 406, 419, 615
MEP 14
MIME メッセージ
  パイプライン構成 138
mime_options
  パイプライン構成エレメント 144
minOccurs
  JSON スキーマ内 324, 413, 517
  XML スキーマ内で 389, 406, 419, 615
mtom
  パイプライン構成エレメント 138
MTOM/XOP
  パイプライン構成 138
mtom_options
  パイプライン構成エレメント 140

N

named_transport_entry
  パイプライン構成エレメント 115
namespace
  パイプライン構成エレメント 119

P

PL/I
  JSON スキーマへのマッピング 500, 506, 598, 604
  XML スキーマへのマッピング 376, 381
PL/I へのマッピング 376, 381, 500, 506, 598, 604
provider_pipeline
  パイプライン構成エレメント 116

R

RACF の構成 667
requester_pipeline
  エレメント、パイプライン定義の 112
  パイプライン構成エレメント 119

S

SAML
  コンテナー 206
Security Token Service
  Trust クライアント・インターフェース 662
service_handler_list
  パイプライン構成エレメント 135
service_parameter_list
  サービス・パラメーター・リスト 136
sign_body
  パイプライン構成エレメント 158
SOAP
  エンベロープ 22
  概説 17
  障害 22
  ヘッダー 22
  本体 22
  SOAP の概要 17
SOAP 障害 635
SOAP パイプライン 65
SOAP メッセージ
  暗号化 665
  構造 22
  署名 663
  例 22
  XML スキーマ
    SOAP メッセージの検証 650
  XML スキーマとの照合 650
SOAP メッセージの検証 650
SOAP メッセージ・パス 9
sts_authentication
  パイプライン構成エレメント 153
sts_endpoint
  パイプライン構成エレメント 157

T

terminal_handler
  パイプライン構成エレメント 117
TRANSFORM DATATOXML 449
TRANSFORM XMLTODATA 450
transport
  パイプライン構成エレメント 137
transport_handler_list
  パイプライン構成エレメント 118
Trust クライアント
  インターフェース 662
  起動 677

U

URI
  WebSphere MQ トランスポートの 56
URI の指定変更 219

W

Web Services Addressing
  アドレッシング・ハンドラー 246, 247
  サポート 241
  仕様 241
  デフォルト EPR 249, 251

Web Services Addressing (続き)
デフォルトのアクション 249, 252, 254
パラメーター 572
プロバイダー・パイプライン構成 247
明示的アクション 249, 252
リクエスター・サービス 249
リクエスター・パイプライン構成 246
DFHWSADH 246, 247
DFHWS-URI 243
EPR 243
MAP 243
WSADDR-EPR-ANY 572
WSDL 1.1 252
WSDL 2.0 254
<wsa:Action> 252
Web Services Security
パイプライン構成 144
Web Services Security (WSS) 655, 667, 672
Web サービスのエラー 684, 686
Web サービスのためのセキュリティー 655
Web サービス・アシスタント 535
サービス・プロバイダー・アプリケーションの作成 315, 628
web サービス・ディスカバリー 51
WSDL
およびアプリケーション・データ構造 12
言語構造への変換 282, 297, 342, 553
照会 51
Web Services Addressing 249
WSDL 1.1
デフォルト EPR 251
WSDL 文書
可変長 411, 436, 620
空白 411, 436, 620
wsse_handler
パイプライン構成エレメント 145
WS-Addressing
アドレッシング・ハンドラー 246, 247
仕様 241
デフォルト EPR 251
デフォルトのアクション 252, 254
パラメーター 572
プロバイダー・パイプライン構成 247
明示的アクション 252
リクエスター・パイプライン構成 246
DFHWSADH 246, 247
DFHWS-URI 243
EPR 243
MAP 243
WSADDR-EPR-ANY 572
WSDL 1.1 252
WSDL 2.0 254
<wsa:Action> 252
WS-AT 221

X

XML
アシスタント 332
解析 441
照会 440
データ型 441
データへ変換 332
変換 441
xsd:any データ型 443
XML アシスタント
DFHLS2SC 445
XML 支援機能 333
XML スキーマ 357, 364, 369, 372, 376, 381
可変長 411, 436, 620
空白 411, 436, 620
XML の解析 441
XML の照会 440
XML の変換 441
XML バインディング 445
XML への言語構造のマッピング 445
xop_options
パイプライン構成エレメント 142
xsd:any データ型 443

Z

zAAP 65
zosConnectService 84, 90
z/OS Connect 46, 84, 88, 90, 94
概説 40
構成 89, 96
セキュリティー 678

[特殊文字]

<addressing>
パイプライン構成エレメント 119
<apphandler>
パイプライン構成エレメント 116
<apphandler_class>
パイプライン構成エレメント 114
<authentication>
パイプライン構成エレメント 147
<auth_token_type>
パイプライン構成エレメント 156
<cics_json_handler_java>
パイプライン構成エレメント 120
<cics_mtom_handler>
パイプライン構成エレメント 139
<cics_soap_1.1_handler>
パイプライン構成エレメント 121
<cics_soap_1.1_handler_java>
パイプライン構成エレメント 123
<cics_soap_1.2_handler>
パイプライン構成エレメント 126
<cics_soap_1.2_handler_java>
パイプライン構成エレメント 128
<default_http_transport_handler_list>
パイプライン構成エレメント 131
<default_mq_transport_handler_list>
パイプライン構成エレメント 131
<default_target>
パイプライン構成エレメント 137
<default_transport_handler_list>
パイプライン構成エレメント 132
<dfhmtom_configuration>
パイプライン構成エレメント 140
<dfhwsse_configuration>
パイプライン構成エレメント 146
<encrypt_body>
パイプライン構成エレメント 159
<handler>
パイプライン構成エレメント 132
<jvmserver>
パイプライン構成エレメント 133
<mime_options>
パイプライン構成エレメント 144
<mtom>
パイプライン構成エレメント 138
<mtom_options>
パイプライン構成エレメント 140
<named_transport_entry>
パイプライン構成エレメント 115
<namespace>
パイプライン構成エレメント 119
<provider_pipeline>
パイプライン構成エレメント 116
<provider_pipeline_json>
パイプライン構成エレメント 116
<repository>
パイプライン構成エレメント 134
<requester_pipeline>
パイプライン構成エレメント 119
<service>
パイプライン構成エレメント 134
<service_handler_list>
パイプライン構成エレメント 135
<service_parameter_list>
パイプライン構成エレメント 136
<sign_body>
パイプライン構成エレメント 158
<sts_authentication>
パイプライン構成エレメント 153
<sts_endpoint>
パイプライン構成エレメント 157
<terminal_handler>
パイプライン構成エレメント 117

<transport>
パイプライン構成エレメント 137
<transport_handler_list>
パイプライン構成エレメント 118
<wsse_handler>
パイプライン構成エレメント 145
<xop_options>
パイプライン構成エレメント 142

日本アイ・ビー・エム株式会社
〒103-8510 東京都中央区日本橋箱崎町19-21