Referência da Linguagem ESDL

Equipe de documentação de Boca Raton

Referência da Linguagem ESDL

Equipe de documentação de Boca Raton

Sua opinião e comentários sobre este documento são muito bem-vindos e podem ser enviados por e-mail para <docfeedback@hpccsystems.com>

Inclua a frase **Feedback sobre documentação** na linha de assunto e indique o nome do documento, o número das páginas e número da versão atual no corpo da mensagem.

2023 Version 9.0.6-1

Visão Geral da Linguagem ESDL

ESDL (Enterprise Service Description Language) é uma metodologia que ajuda você a desenvolver e gerenciar interfaces de consulta baseada na web de forma rápida e consistente.

A ESDL dinâmica adota uma abordagem de desenvolvimento da infertace em primeiro lugar. Ela aproveita a linguagem ESDL para criar um contrato de interface comum de modo que ambos os desenvolvedores de consultas Roxie e de interface Web possam aderir. O objetivo é permitir que os desenvolvedores criem serviços Web de produção com interfaces limpas que capazes de evoluir e crescer ao longo do tempo sem interromper as aplicações existentes.

O suporte de versões embutido do ESDL ajuda a garantir que aplicações implementadas e compiladas continuem a operar mesmo com mudanças na interface do serviço implementado para novas funcionalidades.

A capacidade do ESDL de definir e reutilizar estruturas comuns ajuda a manter as interfaces consistentes entre os métodos.

O serviço ESDL dinâmico foi desenvolvido para escalar horizontalmente, e webhooks são fornecidos para adicionar processos de registro (logging) e segurança personalizados para ajudar a criar serviços Web totalmente "prontos para produção".

Depois que um serviço é implementado, desenvolvedores de aplicações e usuários finais podem utilizá-lo usando REST, JSON, XML, SOAP ou postagens codificadas por formulário. A ESDL dinâmica oferece acesso rápido e fácil a um WSDL, formulários dinâmicos, amostras de busca e de respostas, e interfaces de testes para permitir que os desenvolvedores testem as mudanças lógicas, as alterações de dados ou os novos recursos, além de interagirem com o serviço diretamente usando SOAP, XML ou JSON.

Estruturas ESDL

ESPstruct

ESPstruct é um conjunto de elementos agrupados sob um único nome. Estes elementos, conhecidos como *members*, podem ser de tipos e comprimentos diferentes. As estruturas podem ser aninhadas e são compatíveis com heranças.

Exemplo:

```
ESPstruct NameBlock

{
    string FirstName;
    string MiddleName;
    string LastName;
    int Age;
};
```

ESPrequest

A estrutura de solicitação de um método.

Exemplo:

```
ESPrequest  MyQueryRequest

{
    string FirstName;
    string MiddleName;
    string LastName;
    string Sortby;

    bool Descending(false);
};
```

ESPresponse

A estrutura da resposta de um método.

Exemplo:

```
ESPresponse  MyQueryResponse

{
    string FirstName;
    string MiddleName;
    string LastName;

};
```

ESParray

Uma estrutura para matrizes não vinculadas. As matrizes são compatíveis com heranças e podem ser aninhadas.

Exemplo:

```
ESPstruct NameBlock

{
    string FirstName;
    string MiddleName;
    string LastName;

    int Age;
};

ESParray<ESPstruct NameBlock, Name> Names;
```

Isso resulta em algo parecido com o seguinte:

```
<Names>
  <Name>
    <FirstName>James</FirstName>
    <MiddleName>Joseph</MiddleName>
    <LastName>Deerfield</LastName>
    <Age>42</Age>
  </Name>
  <Name>
    <FirstName>Emily</FirstName>
    <MiddleName>Kate</MiddleName>
    <LastName>Constance</LastName>
    <Age>33</Age>
  </Name>
</Names>
```

ESPenum

Uma estrutura que contém um valor enumerado.

Exemplo:

```
ESPenum EyeColors : string

{
  Brn("Brown"),
  Blu("Blue"),
  Grn("Green"),
};


ESPstruct Person

{
  string FirstName;
  string MiddleName;
  string LastName;

  ESPenum EyeColors EyeColor("Brown"); //provides a default value
};
```

ESPinclude

ESPinclude permite incluir um arquivo ESDL externo. Este comando é semelhante a expressão #incluir.

Exemplo:

```
ESPinclude(commonStructures);
```

ESPservice

Define uma interface de serviço web do ESP. Uma vez estabelecida, essa definição de interface pode ser atribuída (vinculada) a um ESP Service baseado em uma ESDL dinâmica.

Um ESPservice deve conter uma ou mais definições de método.

Exemplo:

```
ESPservice [auth_feature("AllowMyService")] MyService
{
  ESPmethod MyMethod1(MyMethod1Request, MyMethod1Response);
  ESPmethod MyMethod2(MyMethod2Request, MyMethod2Response);
};
```

ESPmethod

Isso estabelece uma definição de método que possa ser referenciada em uma estrutura ESPservice. Esta definição de método deve conter as referências de ESPrequest e ESPresponse estabelecidas previamente.

Exemplo:

```
ESPservice MyService
 {
  ESPmethod MyMethod1(MyMethod1Request, MyMethod1Response);
  ESPmethod
   [
    auth_feature("AllowMyMethod2"),
    description("MyMethod Two"),
    help("This method does everything MyMethod1 does plus a few extra features"),
    min_ver("1.2")
   ]
  MyMethod2(MyMethod2Request, MyMethod2Response);
};
```

Datatypes ESDL

booleanbool

A um tipo de dado lógico ou booleano que possui um ou dois valores possíveis: verdadeiro (1) ou falso (0).

Exemplo:

```
boolean includeFlag;
bool includeMore;
```

string

Um tipo de dados que contém uma sequência de caracteres alfanuméricos.

Exemplo:

```
string FirstName;
```

int

Um valor inteiro

Exemplo:

```
int Age;
```

int64

Um valor inteiro de 64 bits

Exemplo:

```
int64 Iterations;
```

float

Um número real ou de ponto flutuante de 4 bytes..

Exemplo:

```
float Temperature;
```

double

Um número real ou de ponto flutuante de 8 bytes.

Exemplo:

```
double Temperature;
```

binary

Um tipo de dado que contém dados binary (binários), semelhante a um BLOB .

Exemplo:

```
binary RetinaScanSample;
```

ESDL, XSD, e ECL Type Mapping

ESDL	XSD	ECL
Bool boolean	bool	boolean
Binary	Base64Binary	String (base64 encoded)
Float	float	REAL4
Double	double	REAL8
Int	int	INTEGER
Int64	long	INTEGER8
String	string	String

Atributos ESDL

É possível usar atributos ESDL para estender e substituir o comportamento padrão de uma definição ESDL. Por exemplo, a adição de max_len(n) em uma string estabelece que a string precisará armazenar apenas um certo número de caracteres.

Vários atributos são tratados como sugestões que podem, exercer maior influência sobre algumas implementações em relação a outras. Por exemplo, max_len(n) afetará o código ECL gerado, mas é ignorado quando C++ é gerado.

max_len (n)

O atributo *max_len* especifica o comprimento do campo para o campo da string do ECL.

Exemplo:

```
[max_len(20)] string City;
```

No ECL, isso significa que o campo “City” possui um comprimento fixo de 20 caracteres. Para tipo de número inteiro, max_len significa o tamanho máximo em bytes do número inteiro (8*max_len bits integer).

Exemplo:

```
[max_len(3)] int Age;
```

Isso gera o código ECL:

```
integer3 Age{xpath('Age')};
```

Este atributo também funciona para o tipo ESPenum. O tipo ECL também é uma string.

```
[max_len(2)] ESPenum StateCode State;
```

Aqui o StateCode é uma enumeração do código de estado de 2 caracteres.

Este atributo também pode ser usado para ESPstruct, ESPrequest, ESPresponse:

```
ESPstruct [max_len(1867)] IdentitySlim : Identity
  {
     ...
  };
```

Isso gera o código ECL:

```
export t_MyQuery := record (share.t_Name), MAXLENGTH(1867)
  {
  };
```

A opção option *MAXLENGTH* do ECL ajuda o motor ECL a gerenciar melhor a memória.

Isso não afeta o tipo no XSD/WSDL. O ESP ignora este atributo ao gerar o esquema. O tipo para a string é xsd:string, o qual não possui limite de comprimento. Consequentemente, o esquema permanece o mesmo se houver alteração no comprimento do campo na consulta Roxie.

ecl_max_len (n)

Este atributo *ecl_max_len* pede para o gerador ECL usar o ECL *maxlength* em vez do comprimento padrão do campo.

Exemplo:

```
[ecl_max_len(50)] string CompanyName;
[max_len(6)] string Gender;
```

O código ECL gerado é:

```
string CompanyName { xpath("CompanyName"),maxlength(50) };
   string6 Gender { xpath("Gender") };
```

Observação: quando os dois atributos *max_len* e *ecl_max_len* estão especificados, *ecl_max_len* será usado e o *max_len* ignorado.

ecl_name ("name")

O atributo *ecl_name* especifica o nome do campo no código ECL gerado. Por padrão, o nome do campo no ECL é o mesmo nome definido no ECM. No entanto, em alguns casos, o nome poderia causar problemas no ECL. Por exemplo, as palavras-chave no ECL não podem ser usadas como um nome de campo.

Exemplo:

```
[ecl_name("_export")] string Export;
 [ecl_name("_type")] string Type;
```

Aqui, **EXPORT** e **TYPE** são palavras-chave do ECL, portanto não podem ser usadas como nomes de campo do ECL. Usamos *ecl_name* para solicitar para o processo esdl2ecl gerar nomes aceitáveis.

counter e count_val

Estes dois atributos são usados para ajudar o ESP a calcular a contagem dos registros da resposta.

O counter conta o número de registros secundários dos nós. Quando o nó corresponde a uma matriz, o valor é o mesmo do que o número de itens na matriz.

O count_val usará o valor do nó como número de registros. O campo **RecordCount** está implicitamente marcado como *count_val*.

Quando uma resposta possui múltiplos contadores, count_val, a soma dos valores é retornada como número de registros.

Exemplo:

```
[counter] ESParray<MyRecord, Record> Records;
[count_val] int TotalFound;
```

max_count_var

O atributo max_count_var é usado para especificar a quantidade máxima de itens em um dataset (ESParray).

Exemplo:

```
[max_count_var("iesp.Constants.JD.MaxRecords")] ESParray <ESPstruct MYRecord, Record> Records;
```

O Roxie definirá a constante iesp.Constants.JD.MaxRecords e realizará alterações de maneira independente, sem afetar o ESP.

ecl_null (n | string)

O atributo *ecl_null* fala para o ESP remover completamente o campo se o valor de campo for *n* ou *string*. Isso oferece um meio de remover completamente um campo quando não houver nenhum dado para ele.

Exemplo:

```
[ecl_null(0)] int Age;
[ecl_null("false")] bool IsMatch;
```

Age 0 significa que não há dados Age para esta pessoa. Assim, se Age for 0, a tag <Age> não é retornada.

Sem este atributo, **<Age>0</Age>** seria retornada.

No segundo exemplo, o valor booleano de *false*, que retornou como uma string, é considerado como nulo e, consequentemente, a tag não é retornada.

leading_zero(n)

O atributo *leading_zero* adiciona zero(s) ao valor do campo para que o comprimento total seja *n*.

Exemplo:

```
ESPstruct Date { [leading_zero(4)] Year; [leading_zero(2)] Month; [leading_zero(2)] Day; };
```

Assim, a data sempre será formada pelo ano com 4 dígitos e pelo mês e dia com 2 dígitos.

ecl_hide

O atributo *ecl_hide* oculta o campo do ECL (ou seja, o campo é removido quando o código ECL é gerado). Este atributo é usado em alguns casos especiais.

Exemplo:

```
ESPstruct Date
 {
    [leading_zero(4)] Year;
    [leading_zero(2)] Month;
    [leading_zero(2)] Day;
 };
```

Neste caso, a estrutura Relative é definida recursivamente, e o ECL não é compatível com tal estrutura. Por isso, usamos *ecl_hide* para evitar uma definição recursiva no ECL.

Às vezes, um campo é ocultado do ECL por outros motivos. Nestes casos, o atributo *ecl_hide* não é necessário.

ecl_type ("type")

O atributo *ecl_type* define o tipo de campo no ECL.

Exemplo:

```
[ecl_type("Decimal10_2")] double RetailPrice;
```

ESDL não possui um tipo monetário, por isso usamos *ecl_type* para defini-lo.

ecl_keep

O atributo *ecl_keep* mantém o campo no ECL gerado, mesmo que este campo tivesse sido ocultado sem este atributo.

min_ver

O atributo min_ver permite definir a versão mínima (mais antiga) em que um campo possa ser visível. As solicitações que usarem uma versão anterior não terão acesso ao campo.

Exemplo:

```
[min_ver("1.03")] bool IsValid;
```

max_ver

O atributo max_ver permite definir a versão máxima (mais recente) em que um campo possa ser visível. As solicitações que usarem uma versão mais recente não terão acesso ao campo.

Exemplo:

```
[max_ver("1.04")] bool IsValid;
```

depr_ver

O atributo depr_ver permite declarar a versão do final da vida de um campo. O campo é preterido em um número de versão especificado. As solicitações que usam essa versão ou qualquer versão subsequente não terão acesso ao campo.

Exemplo:

```
[depr_ver("1.04")] bool IsValid;
```

get_data_from

O atributo get_data_from permite especificar que um campo obtém seus dados a partir de uma outra variável Isso oferece compatibilidade com a reutilização do código quando são realizadas alterações complexas de versão.

Exemplo:

```
ESPresponse RoxieEchoPersonInfoResponse
{
  ESPstruct NameInfo Name;
  string Var1;
  [get_data_from("Var1")] string Var2;
};
```

No exemplo acima, a consulta retorna os dados em Var1 e, em seguida, o serviço coloca os dados no campo Var2 e envia em resposta para o cliente.

Neste exemplo, tanto Var1 quanto Var2 estão na resposta para o cliente. Normalmente, Var1 e Var2 fazem parte das versões sem sobreposição, por isso apenas uma delas estará na resposta (dependendo da versão especificada).

Uma vez que o atributo get_data_from é compatível com tipos de dados complexos, como um ESPstruct, os campos não precisam ser limitados aos tipos de string.

optional

O atributo optional permite especificar que um campo é opcional e oculto, ou que não depende da presença ou ausência de uma decoração de URL

Quando um campo possui um atributo optional , ele fica visível apenas quando a opção aparece no URL. Porém, quando a opção começa com um ponto de exclamação (!), o campo é ocultado apenas se a opção estiver no URL.

Exemplo:

```
ESPrequest RoxieEchoPersonInfoRequest
{
     ESPstruct NameInfo Name;
                            string First;
                            string Middle;
                            string Last;
     [optional("dev")]      string NickName;
     [optional("!_NonUS_")] string SSN;
};
```

Supondo que o serviço esteja sendo executado em um servidor com nome de host example.com e que a ligação do serviço esteja definida para 8003:

Se a URL for

```
http://example.com:8003/
```

, o SSN estará visível e o NickName ocultado;

Se a URL for

```
http://example.com:8003/?dev
```

, o SSN e o NickName estarão visíveis

Se a URL for

```
http://example.com:8003/?dev&_NonUS_
```

, o NickName estará visível e o SSN ocultado.

Se a URL for

```
http://example.com:8003/?_NonUS_
```

, o NickName e o SSN estarão ocultados.

help

O atributo help (válido apenas para o ESPMethod) permite especificar alguns textos adicionais a serem exibidos no formato em que são gerados automaticamente para executar um método.

Exemplo:

```
ESPservice MyService

 {
   ESPmethod MyMethod1(MyMethod1Request, MyMethod1Response);
   ESPmethod
    [
     description("MyMethod Two"),
     help("This method does everything MyMethod1 does plus a few extra features"),
     min_ver("1.2")
    ]
   MyMethod2(MyMethod2Request, MyMethod2Response);
 };
```

description

O atributo description (válido apenas para o ESPMethod) permite especificar alguns textos adicionais a serem exibidos no formato em que são gerados automaticamente para executar um método.

Exemplo:

```
SPservice MyService
{
  ESPmethod MyMethod1(MyMethod1Request, MyMethod1Response);
  ESPmethod
   [
    description("MyMethod Two"),
    help("This method does everything MyMethod1 does plus a few extra features"),
    min_ver("1.2")
   ]
  MyMethod2(MyMethod2Request, MyMethod2Response);
};
```

version e default_client_version

Os atributo **version** e **default_client_version** (válidos apenas em ESPService) permitem especificar a versão a ser usada quando uma versão não for especificada de maneira explícita na solicitação.

O atributo **default_client_version** é usado para solicitações de API no formato SOAP, se o cliente não especificar a versão. O atributo **version** é usado para solicitações originadas de um navegador da Internet sem uma decoração de versão no URL.

Esses atributos fornecem melhor compatibilidade com versões anteriores do API e permitem que os desenvolvedores de API vejam a interface mais recente usando um navegador.

Se **o atributo default_client_version** for superior ao **version**, o serviço usará o atributo **default_client_version** para todas as solicitações que não especificarem o atributo **version**.

Mesmo que padrões possam ser especificados para um serviço, os desenvolvedores de API devem ser incentivados a especificar uma versão nas solicitações a fim de garantir a compatibilidade entre a aplicação e o serviço.

Exemplo:

```
 ESPservice [version("0.02"), default_client_version("0.01")] EsdlExample
{
    ESPmethod EchoPersonInfo(EchoPersonInfoRequest, EchoPersonInfoResponse);
    ESPmethod RoxieEchoPersonInfo(RoxieEchoPersonInfoRequest, RoxieEchoPersonInfoResponse);
};
```

auth_feature

O atributo auth_feature (válido apenas para ESPService ou ESPMethod) permite especificar meios de verificar as permissões de um usuário para executar um método.

Para ativar este recurso, seu sistema deve estar configurado para usar um formulário de segurança compatível com a autenticação em nível de recurso, tal como a segurança LDAP incluída na Community Edition da plataforma. Depois da LDAP ter sido configurada, adicione o nome da tag fornecido como o valor do atributo **authFeature** à lista de autenticação em nível de recurso na seção “Segurança” do ECL Watch. Em seguida, defina as permissões para os usuários e/ou grupos.

Se estiver usando um Security Manager Plugin (Plugin do Gerenciador de Segurança) de terceiros, consulte a documentação do plugin para obter detalhes sobre como adicionar nome de tag à configuração de segurança.

O atributo auth_feature é obrigatório para cada método, mas pode ser especificado no nível ESPService para ser aplicado a todos os métodos em um serviço. Você pode substituir um método individual configurando o atributo em um nível de método

A configuração auth_feature("NONE") significa que nenhuma autenticação é necessária. A configuração auth_feature("DEFERRED") adia a autenticação para a lógica de negócios na lógica de implementação do método do desenvolvedor do ESP .

Exemplo:

```
ESPservice MyService [auth_feature("NONE")]
{
  ESPmethod MyMethod1(MyMethod1Request, MyMethod1Response);
  ESPmethod
   [
    description("MyMethod Two"),
    auth_feature("AllowMyMethod2"),
    help("This method does everything MyMethod1 does plus a few extra features"),
    min_ver("1.2")
   ]
  MyMethod2(MyMethod2Request, MyMethod2Response);
};
```