Portrait
Foundation

Host Integration Framework User Guide

Edition 1.0

30 June 2014

Pitney Bowes
Software

Portrait Foundation
Host Integration Framework User Guide

Acknowledgement of trademarks

Other product names, company names, marks, logos and symbols referenced herein may be the trademarks or registered trademarks of their registered owners.

About Portrait Software

Portrait Software is now part of Pitney Bowes Software Inc.

Portrait Software enables organizations to engage with each of their customers as individuals, resulting in improved customer profitability, increased retention, reduced risk, and outstanding customer experiences. This is achieved through a suite of innovative, insight-driven applications which empower organizations to create enduring one-to-one relationships with their customers.

Portrait Software was acquired in July 2010 by Pitney Bowes to build on the broad range of capabilities at Pitney Bowes Software for helping organizations acquire, serve and grow their customer relationships more effectively. The Portrait Customer Interaction Suite combines world leading customer analytics, powerful inbound and outbound campaign management, and best-in-class business process integration to deliver real-time customer interactions that communicate precisely the right message through the right channel, at the right time.

Our 300 + customers include industry-leading organizations in customer-intensive sectors. They include 3, AAA, Bank of Tokyo Mitsubishi, Dell, Fiserv Bank Solutions, Lloyds Banking Group, Merrill Lynch, Nationwide Building Society, RACQ, RAC WA, Telenor, Tesco Bank, T-Mobile, Tryg and US Bank.

Pitney Bowes Software Inc. is a division of Pitney Bowes Inc. (NYSE: PBI).

For more information please visit: http://www.pitneybowes.co.uk/software/

UK

Portrait Software
The Smith Centre
The Fairmile
Henley-on-Thames
Oxfordshire, RG9 6AB, UK

Email: support@portraitsoftware.com
Tel: +44 (0)1491 416778
Fax: +44 (0)1491 416601

America

Portrait Software
125 Summer Street
16th Floor
Boston, MA 02110
USA

Email: support@portraitsoftware.com
Tel: +1 617 457 5200
Fax: +1 617 457 5299

Norway

Portrait Software
Portrait Million Handshakes AS
Maridalsveien. 87
0461 Oslo
Norway

Email: support@portraitsoftware.com
Tel: +47 22 38 91 00
Fax: +47 23 40 94 99

About this document

Purpose of document

The Host Integration Framework is a flexible mechanism for integrating Portrait Foundation with external systems. It provides an environment where various components (Transformers, Adapters, Data Access Transactions and Mapping Editors) can be plugged in as required. This document describes the purpose of these components in detail.

Intended audience

Portrait Foundation configurers and developers who want to make use of Host Integration Framework to integrate to external systems.

Related documents

Extending Applications using .NET *(Portrait NET SDK)*

Software release

Portrait Foundation 5.0 or later.

Contents

1 Host Integration Framework

1.1 Rationale

The Host Integration framework was introduced to address several issues related to interactions between Portrait Foundation and a host system. In a typical project implementation scenario it is necessary to integrate Portrait Foundation with existing host systems such as legacy mainframes or existing Web Services.

Traditionally, the Data Access Transaction component was responsible for the entire task of host system communication. For example, every single standard Portrait Foundation transaction accessing the Portrait Foundation database must cope with all the details of interactions with the SQL Server. This approach offers various benefits, such as full control over the process and its performance. However, its main drawback is lack of reuse and ease of configuration.

The Host Integration framework extends existing data access capabilities by providing a set of reusable and configurable components together with an environment in which they efficiently interact. There are three major types of components:

1.1.1 Data Access Transaction component

A data access transaction component was traditionally responsible for the entire task of a host system interaction. While it is still possible to write ad-hoc Data Access Transaction components, the preferred method is to use the new Generic Data Access Transaction component. The Generic DAT is an entry point to the framework functionality that enables reuse and configurability.

1.1.2 Transformer component

A transformer component's responsibility is to transform input data from Portrait Foundation format into that required by a host system request, or to transform the host system response back to Portrait Foundation format. It is possible to chain the components together in order to reuse existing transformers. For example we use one transformer to convert Portrait Foundation data into XML and then use another transformer to transform XML into an XML format required by a host system.

1.1.3 Adapter component

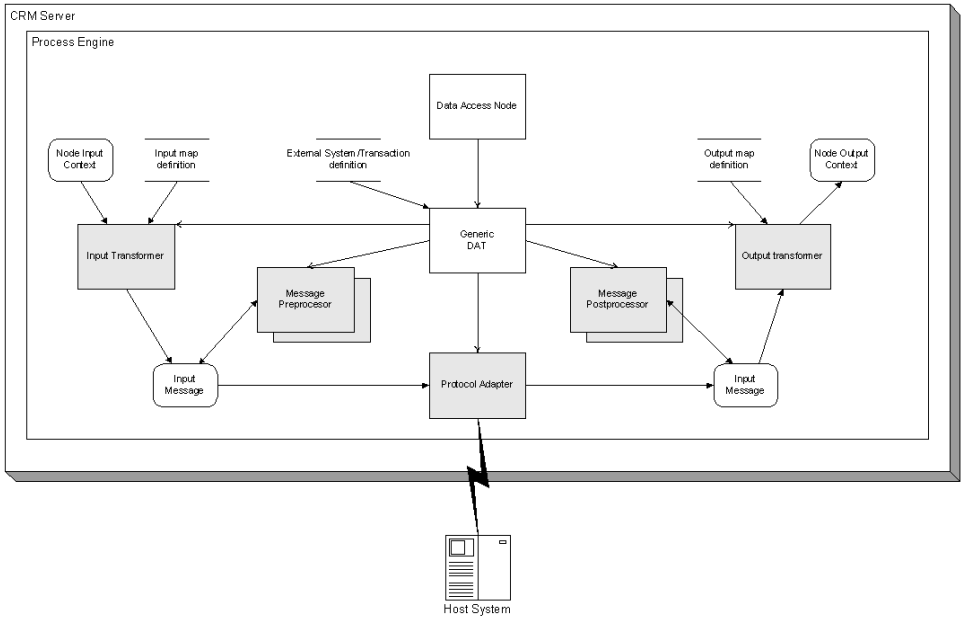
An adapter component is responsible for the actual transmission of data to and from a host system. Various components are available in order to communicate via number of channels such as XML over HTTP or MSMQ. In addition, due to the ability to connect a number of adapters together we can easily create a stack of protocols reflecting a host system's needs.

By combining and configuring the components above we can create powerful and flexible host integration solutions. The Host Integration framework provides an opportunity to decouple host specific data from host specific transmission. In addition, the high configurability of the framework allows the interchange of existing components and introduction of new ones with minimal impact on existing solutions. For example when a client changes the specification of the input data we simply amend the configuration of our transformers but the host system protocol stack stays intact.

1.2 How it works

1.2.1 The runtime components

Figure 1 – Host Integration components at runtime.



Data Access Node

The Data Access Node within a model is configured to execute a transaction of a particular system. The node instantiates a component with the CLSID defined in the transaction property dialog.

In the diagram, the node creates an instance of the Generic DAT component.

Generic DAT

The Generic DAT component loads the deployed transaction configuration and creates instances of transformer and adapter components. It is its responsibility to orchestrate the entire transformer – adapter scenario.

Transformer component

A transformer transforms input message to output message. The transformation definition (input/output map definition) is part of transformer configuration. For example, in the case of an XSLT transformer, the transformation is stored as a string containing XSLT.

Input/Output message

A message simply contains the data. Portrait Foundation does not specify the type or format of the data. The only requirement is that it must be possible to store the message in a variable of type VARIANT.

Portrait Foundation does not check for message incompatibilities. A project team configuring the transactions must make sure that configured transformers and adapters are compatible.

Adapter component

An adapter component is responsible for direct communication with a host system. It is possible to create a stack of adapters and split the complexity of host integration into several adapters.

Adapter may be supplied with DAT-specific configuration data.

Synchronisation component (not shown)

There are situations where a host system does not return a result immediately; instead it may return results in a separate response message. The synchronisation component is intended to allow an adapter component to wait for an expected response message.

This component should only be used in a solution where the anticipated wait for responses is very short since a process engine thread will be suspended while the adapter waits.

1.2.2 Generic Data Access Transaction

The Generic Data Access Transaction component is a Data Access Transaction component with a special purpose. Rather than accessing a database directly, the component uses the transformers and adapters to do the work.

The component implements **IAmcDataAccessTxn2** and uses the Transaction Helper component to obtain information about the transformers and adapters configured for a particular transaction.

The main tasks of the component are:

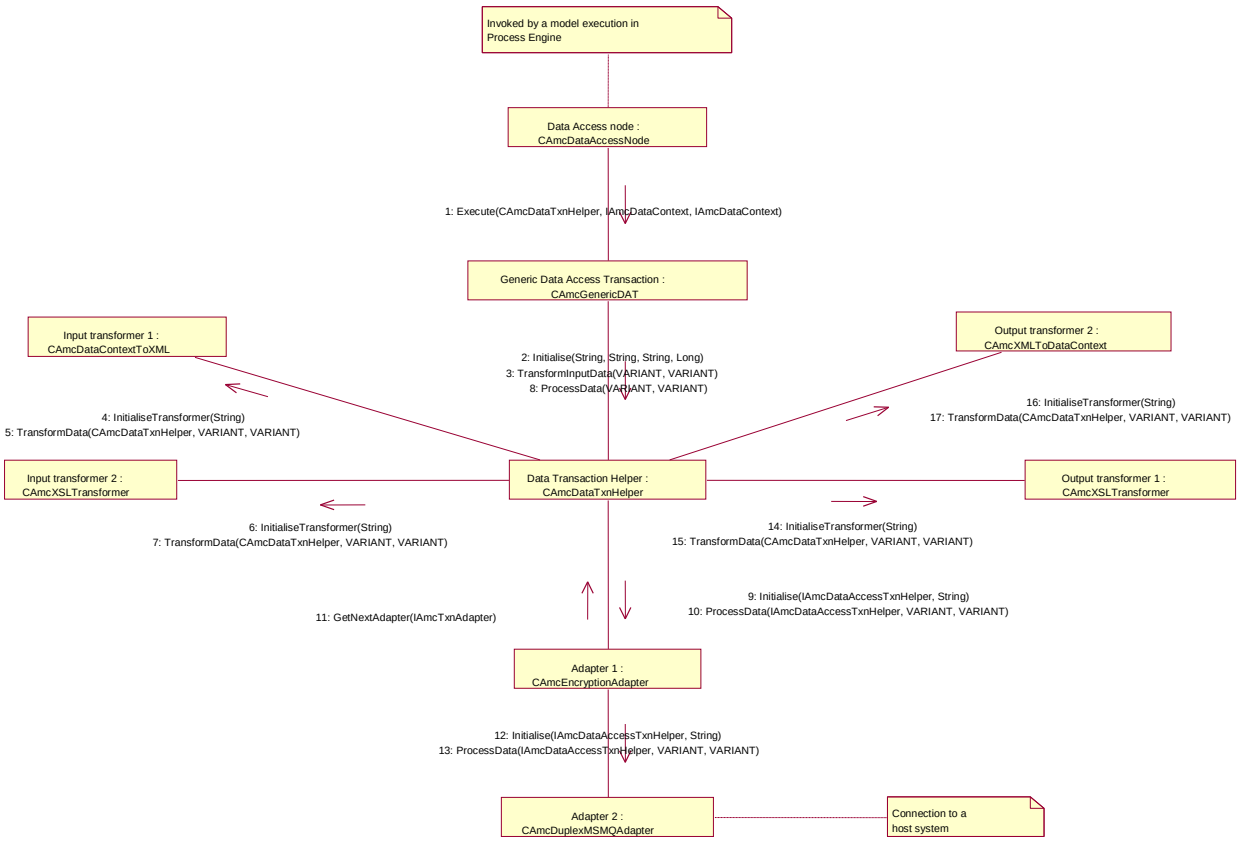
- Apply input transformers to the input data
- Call the first adapter in the stack to process the transformed data
- Apply output transformers to the result returned from the adapter
- The Generic DAT is the entry point to any transformer/adapter interaction, however, it has no special status and is treated is in the same way as any other transaction component.
- The only difference between ordinary and host system transactions (involving transformer/adapter execution) is that the latter is configured to use the Generic DAT CLSID.
- In addition, it is possible to write a custom transaction with the same functionality as the Generic DAT using Foundation SDK.

Runtime requirements

- The Generic DAT passes the Data Context to the first input transformer and expects the last output transformer to return a Data Context.
- If there are no input/output transformers then an adapter must be capable of receiving/returning a Data Context.
- For example, when we use XML to communicate with a host system then the first input transformer would be *Data Context to XML* and the last output transformer would be *XML to Data Context*.
- If there is no adapter, the output of the last input transformer is passed as input to the first output transformer.
- In order to return details of runtime errors, transformer and adapter components should return a Data Context containing a single Data Object with the system name **ErrorDetail**. If a transformer or adapter fails but does not return an **ErrorDetail** Data Object, the Generic DAT will create one.

1.2.3 Runtime components interaction

Figure 2 – Host Integration components interaction when sending XML over MSMQ.



- The diagram above shows interaction between host integration components when sending and receiving encrypted XML via MSMQ.

1.2.4 Host Integration Transformer

The main responsibility of a transformer component is to transform the data on the way to a host system, or on the way back. The transformer components play an important role in almost any host integration scenario as it is very unlikely that two different host systems would use identical data formats.

The IAmcTxnTransformer interface

```
[propget]
HRESULT bStateless(
    [out, retval] VARIANT_BOOL *pbStateless
);

HRESULT Initialise(
    [in] IAmcDataAccessTxnHelper *pTxnHelper,
    [in] BSTR bstrTransformationData
);

HRESULT TransformData(
    [in] IAmcDataAccessTxnHelper *pTxnHelper,
    [in] VARIANT vInputData,
    [out, retval] VARIANT *pvOutputData
);
```

The **Initialise()** method is called by Portrait Foundation after an instance of the component is created. The **bstrTransformationData** parameter contains the configuration time data specified in the Configuration Suite. For example, the XSLT transformer would be configured with a XSL transformation and the transformation would appear in **bstrTransformationData** at runtime.

In order to capture necessary configuration time information the adapter may need to implement a special transformer editor component. We discuss the development of such components in the next chapter.

The **TransformData()** method is called when the data needs to be transformed.

Data format requirements

The Host Integration framework does not specify what the type of an input or output of a transformer must be. The requirement is that it must be possible to pass the data around as a VARIANT.

It is the responsibility of the transformers and adapters to make sure that data stored as VARIANT are in the correct format and the entire configured host integration scenario is type compatible.

The data passed from and to the Generic DAT must be a data context. A data context is a COM object and a pointer to its IAmcDataContext interface is stored in a VARIANT. As the **IAmcDataContext** interface is an **Idispatch** interface, the type of a VARIANT is VT_DISPATCH.

This means that if a transformer is expected to talk directly to the Generic DAT it must be capable of accepting/returning a data context object. For example, the Data Context to XML transformer is capable of receiving a data context object.

Chaining transformers together

It is possible to create a sequence of data transformations by chaining a number of transformers together. In the current Portrait Foundation implementation, there is no compatibility validation that could automatically detect any data type problems.

Transformer lifetime

Some transformers, once initialised, may not maintain any state during the transformation process. It, therefore, may be desirable to cache the transformer object for a particular transaction so that the transformer will not need to be recreated for a different instance of the same type of transaction. This will also allow multiple transactions, of the same type running simultaneously, to use the same transformer object.

If a transformer is performing a truly stateless transformation then it can return a VARIANT_TRUE value for the **bStateless** property, if the transformer maintains

some state during the transformation process then it must return VARIANT_FALSE for the **bStateless** property.

When implementing a new type of transformer it is recommended that you return VARIANT_FALSE for the stateless property, unless you are absolutely sure that the transformer is stateless. If VARIANT_TRUE is returned but the transformer is not stateless, undefined behaviour may, and almost certainly will, occur.

- The XSLT transformer is an example of a stateless transformer as the only shared data is the transformation definition which does not change during the transformer lifetime.

1.2.5 Host Integration Transformer Editor

As we mentioned in the previous section, the main responsibility of a transformer component is to transform the data on the way to and back from a host system. It is very likely that the same transformer needs to be configured to perform various transformations for different transactions. In order to configure a transformer, there must be some user interface component that allows a user to specify the transformer's configuration.

The Host Integration Framework has the concept of a Transformer Editor component that implements the user interface for a particular transformer. It is possible to invoke the user interface from the Configuration Suite and the framework provides a component with access to the entire Portrait Foundation configuration. This means it is possible to create a user interface which is functionally rich and makes the task of configuring a transformer easier.

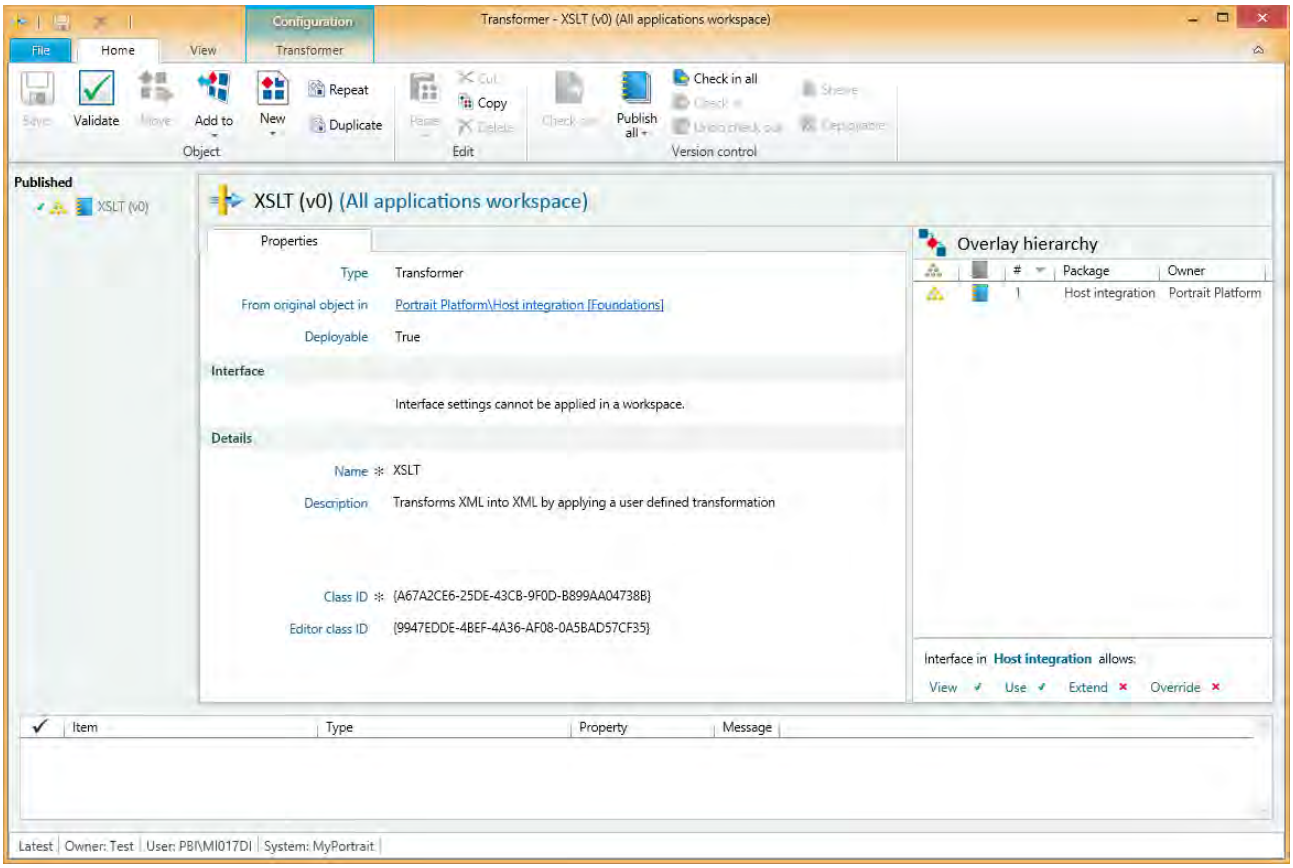
The IAmcTxnTransformerEditor interface

```
HRESULT EditTransformer(
    [in] const long hParentWnd,
    [in] const VARIANT_BOOL bReadonly,
    [in] IAmcConfigHelper *pConfigHelper,
    [in, out] BSTR *pbstrTransformationData
);
```

The **EditTransformer()** method is called by the Configuration Suite when a custom transformer editor needs to be displayed. The **hParentWnd** parameter contains the handle of the Configuration Suite main application window. The **bReadonly** flag is set to VARIANT_TRUE when a transaction is not checked-out and no changes to the transformer configuration are permitted. The **IAmcConfigHelper** pointer to the helper instance provides the editor with a way of accessing the configuration available in the Configuration Suite. The **pbstrTransformationData** contains the actual transformer configuration.

Setting up a transformer editor

Figure 3 – Configuration of the XSLT transformer and its editor component.

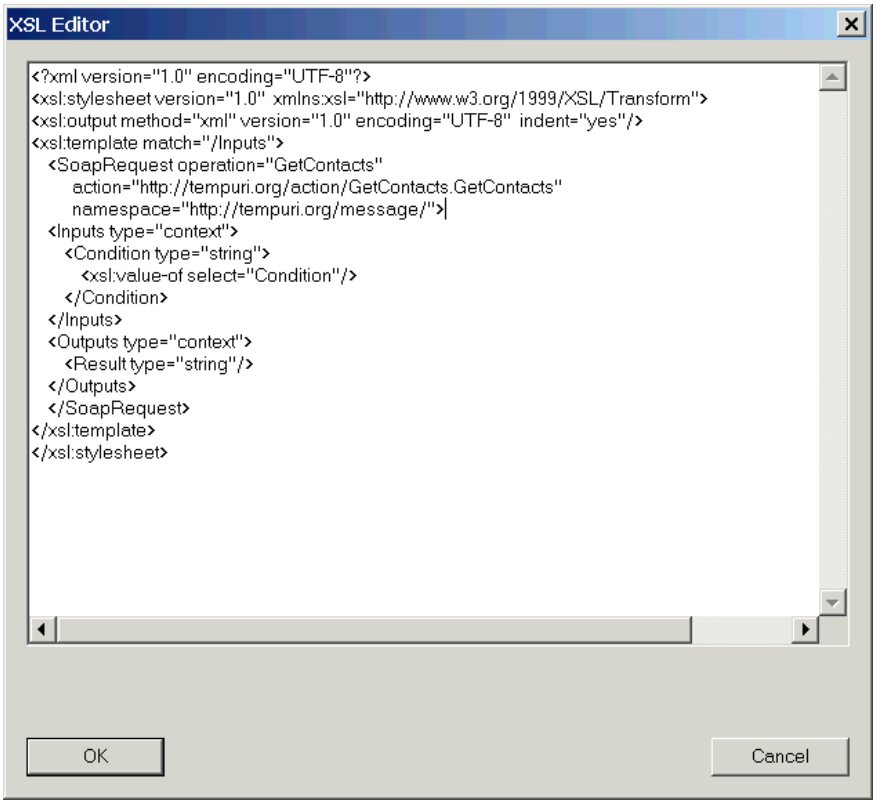


In order to enable a transformer editor to get control over a transformer configuration it is necessary to relate the two with each other. This is done by specifying the transformer editor CLSID in the **Editor Class Id** property of the transformer's property dialog.

As we can see the transformer and the transformer editor components are fully decoupled. This means we could possibly reuse an existing editor implementation to configure a number of different transformers.

Default editor implementation

Figure 4 - XSL transformer editor



- There is a default implementation of a transformer editor component that can be used to enter the transformer's configuration as text. The editor is primarily used when configuring the XSLT transformer. The component `CLSID` is **{9947EDDE-4BEF-4A36-AF08-0A5BAD57CF35}**.

1.2.6 Host Integration Adapter

The main responsibility of an adapter component is to transfer transaction data to a host system, receive a response, and pass it back to the framework for further processing.

The IAmcTxnAdapter interface

```
HRESULT Initialise(
    [in] IAmcDataAccessTxnHelper *pTxnHelper,
    [in] const long nAdapterOrder,
    [in] const long nNumAdapters,
    [in] const BSTR bstrConfigLocation
);

HRESULT ProcessData(
    [in] IAmcDataAccessTxnHelper *pTxnHelper,
    [in] const VARIANT vInputData,
    [out, retval] VARIANT *pvOutputData
);
```

The **Initialise()** method is called by the transaction helper component after an instance of the component is created. The **bstrConfigLocation** parameter contains the value defined in the adapter's property dialog in the Configuration Suite.

The **ProcessData()** method is called when the data needs to be processed.

The IAmcTxnAdapter2 interface

```
HRESULT Initialise(
    [in] IAmcDataAccessTxnHelper *pTxnHelper,
    [in] const long nAdapterOrder,
    [in] const long nNumAdapters,
    [in] const BSTR bstrConfigLocation,
    [in] const BSTR bstrAdapterData
);

HRESULT ProcessData(
    [in] IAmcDataAccessTxnHelper *pTxnHelper,
    [in] const VARIANT vInputData,
    [out, retval] VARIANT *pvOutputData
);
```

It is possible to configure adapter data on a per-DAT basis. The new **IAmcTxnAdapter2** interface, the **Initialise()** method has an additional **bstrAdapterData** parameter. This contains the data configured in the adapter's editor dialog in the Configuration Suite. The Generic DAT will use the new **IAmcTxnAdapter2** interface in preference to the earlier version.

Data format requirements

The Host Integration framework does not specify what the type of an input or output of an adapter must be. The requirement is that it must be possible to pass the data around as a VARIANT.

It is the responsibility of the transformers and adapters to make sure that data stored as VARIANT are in the correct format and the entire configured host integration scenario is type compatible.

The data passed from and to the Generic DAT must be a data context. A data context is a COM object and a pointer to its **IAmcDataContext** interface is stored in a VARIANT. As the **IAmcDataContext** interface is an **Idispatch** interface, the type of the VARIANT is VT_DISPATCH.

This means that if an adapter is expected to talk directly to the Generic DAT it must be capable of accepting/returning a data context object.

Adapter protocol stack

It is possible to chain adapters together to create a protocol stack. Based on where in the stack an adapter can be placed we recognise

Position in the stack	Name	Description
At the bottom	End to End adapter	Direct communication with a host system. May use a high level protocol such as XML over HTTP or MSMQ to send/receive data.
At the top or in the middle	Pass Through adapter	Transforms data on the way to End to End adapter. May be used for encryption, auditing.

As we can see, a Pass Through adapter is very similar to a transformer. Both of them can transform data and sometimes it might be difficult to decide whether to implement certain functionality as a transformer or an adapter. We suggest the following guidelines when making the decision:

- When the transformation needs to be configurable from the Configuration Suite, then consider a transformer.
- When the transformation is business data independent such as encryption then consider an adapter.
- When input and output transformations are related and need to share common knowledge (e.g. it needs to remember the time when the input transformation was started in order to measure total time of execution), then consider an adapter.

Also note that each adapter in the stack may have an associated editor. Per-DAT adapter data is configured separately for each adapter rather than for the stack as a whole.

Adapter lifetime

- Unlike a transformer, an adapter is not cached and it is expected to be stateful. For every transaction there are as many new adapter components created as there are adapters in a stack.

1.2.7 Host Integration Adapter Editor

As we mentioned in the previous section, from the main responsibility of an adapter component is to provide the communication path to and from a host system. It is possible that the same adapter needs to be supplied with some configuration data in order to perform different processing for different transactions. Before Portrait Foundation Release 2.6, this per-DAT configuration data had to be supplied as part of the adapter input message.

In order to configure an adapter, there must be some user interface component that allows a user to specify the adapter's per-DAT configuration.

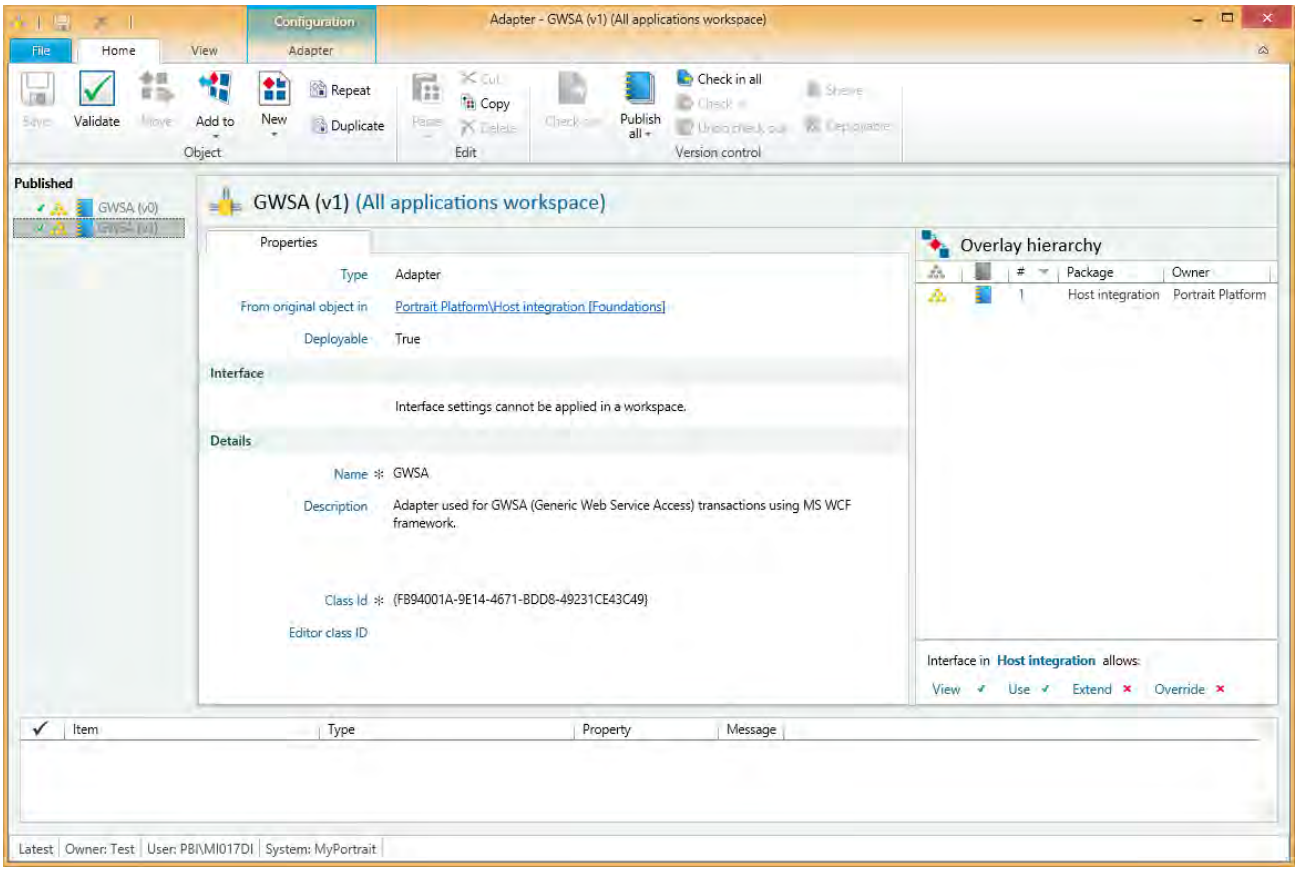
The Host Integration Framework has the concept of an Adapter Editor component that implements a user interface for specifying the adapter's per-DAT configuration for a particular adapter. It is possible to invoke the user interface from the Configuration Suite and the framework provides a component with access to the entire Portrait Foundation configuration. This means it is possible to create a user interface which is functionally rich and makes the task of configuring an adapter easier.

The adapter editor interface

The functionality which must be implemented by an adapter editor is identical to that which must be implemented by a transformer editor. Therefore, for simplicity, the **IAmcTxnTransformerEditor** interface has been reused.

Setting up an adapter editor

Figure 5 - Configuration of the Generic Web Service adapter.



In order to enable an adapter editor it is necessary to associate it with the adapter. This is done by specifying the adapter editor CLSID in the **Editor Class Id** property of the adapter's property dialog. (In the example above there is no editor specified for the chosen adapter.)

As we can see the adapter and the adapter editor components are fully decoupled. This means we could possibly reuse an existing editor implementation to configure a number of different adapters. (In fact, since the interface implemented by adapter editors is the same as that implemented by transformer editors, they could be used interchangeably.)

1.2.8 Synchronisation

The main responsibility of synchronisation component is to allow an adapter to wait for a response from a host system. In order to make use of this component, a service to receive such responses must be implemented as part of the solution e.g. a Web Service.

Component identifiers

```
CLSID: __uuidof(AmcMessageSync)
PROGID: AIT.AMC.HostIntegration.MessageSync
```

The IAmcMessageSync interface

```
HRESULT waitMessage(
    BSTR bstrMessageId,
    enum AMC_MSG_DIRECTION nDirection,
    long nTimeout,
    BOOL* pbMessageArrived,
    BSTR* pbstrMessage
);

HRESULT notifyMessage(
    BSTR bstrMessageID,
    enum AMC_MSG_DIRECTION nDirection,
    BSTR bstrMessage
);
```

When an adapter wishes to wait for an asynchronous response from a host system, it creates an instance of the synchronisation component and calls the **waitMessage()** method.

The service receiving asynchronous responses from the host system must create an instance of the synchronisation component (typically via the Object Broker) and call the **notifyMessage()** method.

The **bstrMessageId** parameter contains the unique identifier of the message. The maximum length is 254 characters.

The **nDirection** parameter is for future use to cater for the possibility that some data should be returned to the host system. Currently a value AMC_MSG_REQUEST must be supplied.

The **nTimeout** parameter indicates how long in milliseconds the adapter wishes to wait. A value of 0 indicates that the adapter does not wish to wait.

The **pbMessageArrived** parameter contains a pointer to a BOOLEAN output value that will be set to TRUE if a message is received within the specified timeout, otherwise a value FALSE will be set.

The **pbstrMessage** parameter contains a pointer to the BSTR into which the received message should be placed.

The **bstrMessage** parameter contains the received message.

Data format requirements

The synchronisation component does not specify what the format of the message should be. See the appropriate adapter's documentation.

1.3 Creating HIF components

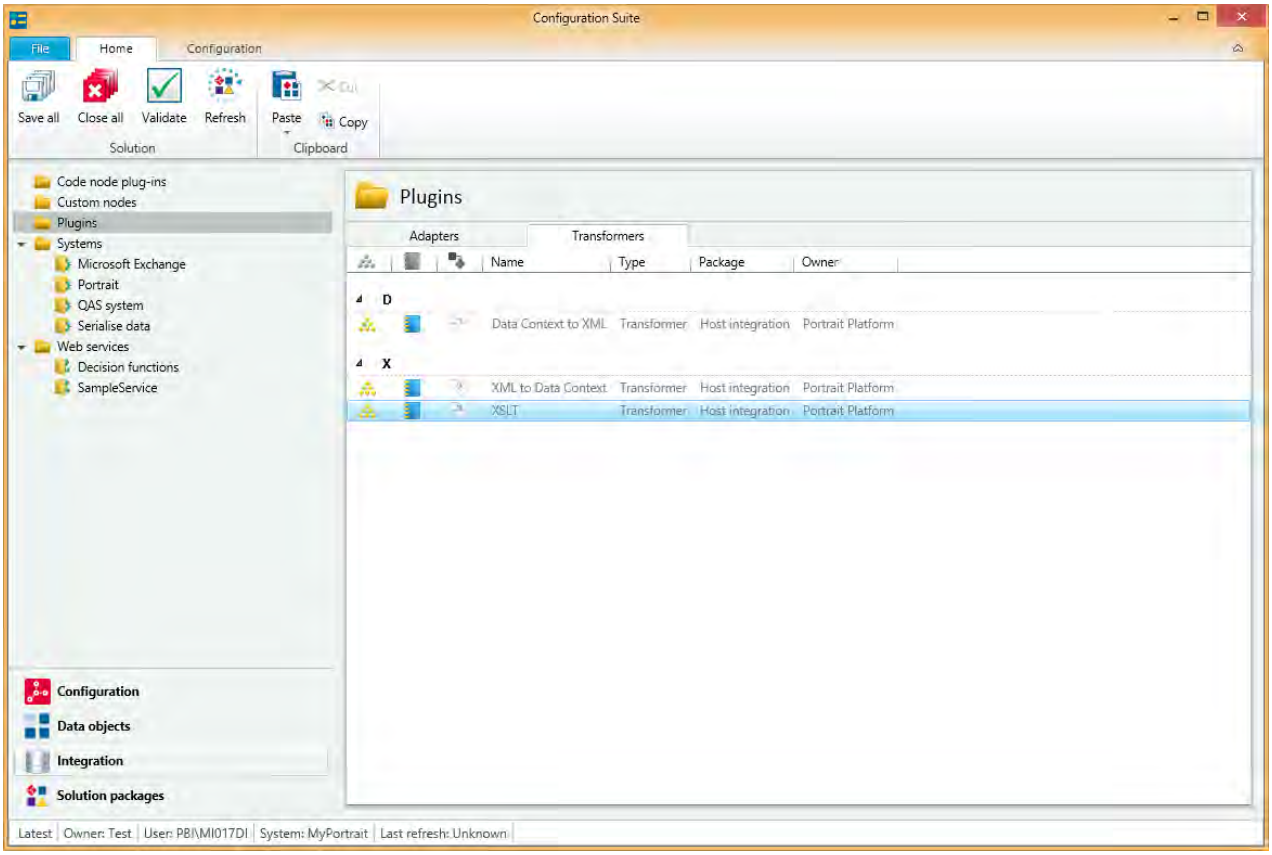
The Portrait Foundation Software Development Kit provides Visual Studio templates for creating the following custom components using C#.

- Adapter
- Transformer
- Transformer Editor

Please refer to the relevant SDK documentation for more details.

1.4 Configuration

Figure 6 – Transaction definitions in Configuration Suite.



Before a transaction can be used in a model it must be defined in the Configuration Suite. All Portrait Foundation specific transactions are defined under a the "Portrait" system. Typically there would be one system definition per host system.

1.4.1 Transaction plug-in components

Transformer and adapter components must be registered before they can be used in transaction definitions. For every component we define **Name**, **Description**, and **CLSID** of the relevant runtime and editor components.

1.4.2 System adapter stack

For every system we can define a stack of adapters that need to be executed when communicating with a host system. For every adapter, we can define its configuration location such as the registry path or the file system path of an INI file.

All transactions within a system share a common adapter stack. The adapter stack definition is deployed to the Portrait Foundation database.

It is possible to have an empty adapter stack as is done in the case of the Portrait Foundation system. In this case the system uses only ordinary transactions that are implemented as self-sufficient components and do not use adapters.

1.4.3 Transaction definition

In order to use the Generic DAT, a transaction must be configured in a specific way:

- The **Class Id** field in Properties dialog must be set to **{C2082E6A-C028-4487-B37F-A1A7797E3A5D}**. This ensures that the Generic DAT component is executed when the transaction node is triggered within a model.
- The transaction must have two outcomes with system names: **OK** and **Fail**.

The Generic DAT has no requirements regarding its inputs or output. They are determined by the requirements of the input and output transformers, if used. Multiple input and output transformers can be added to the transaction definition, with the choice of transformer being made from those listed in the Transaction plug-ins folder. Some transformers can be further configured through the Edit transformer option in the menu (uses an implementation of the Transformer Editor interface).

There is no validation to make sure that configured transformers are compatible with each other.

The transaction definition together with the configured transformers are deployed to Portrait Foundation database.

1.5 Transformer examples

1.5.1 XSLT transformer

The XSLT transformer is a good example of a host integration transformer as it is reasonably simple and still very powerful. The transformer applies a configured XSL transformation to an input XML and returns the result as its output.

When writing the transform, make sure that you test for and include the necessary XSLT code for error handling outputs. Without this, errors are likely to be passed out as a system-level error which will stop the model and generate a critical error dialog.

Input format

The input to the adapter must be a string that contains an XML string or a pointer to an instance of DOM document.

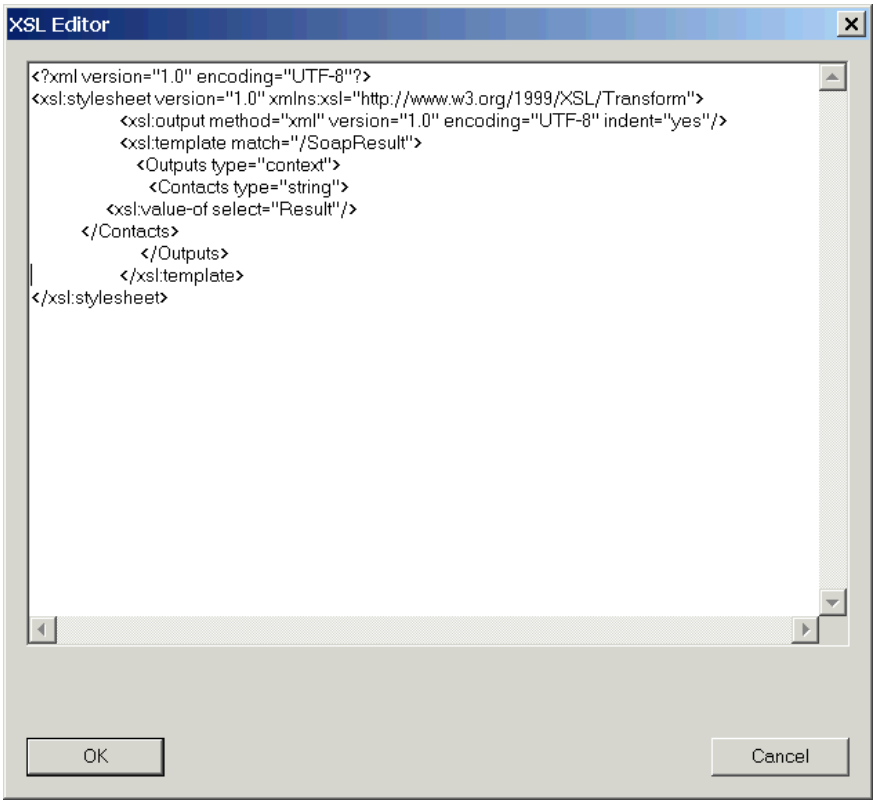
If the input is of type VT_BSTR then the bstrVal property should contain an XML string. If the input is of type VT_UNKNOWN or VT_DISPATCH then **punkVal** should contain a pointer to a DOM document.

Output format

The output of the transformer is always of type VT_BSTR and it contains the result of a XSL transformation. The result may or may not be an XML document.

Transformer configuration

Figure 7 – A transformation definition for the XSLT transformer.



The transformer is configurable using the default transformer editor in which an XSL transformation can be specified.

Implementation

The adapter uses MSXML4 to parse the input XML and apply a transformation. The transformer is implemented as stateless which means that the data access cache component caches the transformer instance.

1.6 Base adapters

1.6.1 XML over HTTP adapter

This component provides low level access to sending and receiving arbitrary XML data over HTTP. No processing of the input data stream is performed prior to transmission or after receipt. As the adapter is data protocol neutral (i.e. it has no expectations or knowledge of the format of data passing through it) it is a flexible mechanism for conversing with endpoints in either proprietary or standard (e.g. Soap) formats. The burden is on the caller (and configurer) to ensure that the resulting request is in a format intelligible to the url that the data is transmitted to. By the same token the caller is responsible for interpreting the returned data.

The adapter supports Portrait Foundation's mechanism for asynchronous adapter calls. Typically these calls are made as pairs. The first call initiating a request and the second call attempting to retrieve a response via an arbitrary **transactionId**.

Although Soap unaware; the adapter is capable of being used for Soap method calls. This support extends to the so-called "Oneway" method calls.

The adapter allows configuration of a timeout value associated with the HTTP request being made. If any call fails because of a timeout the DAT will return a dataobject describing the failure

The adapter uses the **ServerXMLHTTP** object and therefore is not limited by the threading/connection restrictions associated with "client" HTTP objects as it is not built using the WinInet stack.

The configuration, which can be at the HIF System, DAT, or message level can describe attributes of the request e.g. endPointURL, timeouts, and HTTP headers (required in some Soap scenarios)

Configuration

The parameters that control the operation of the adapter can come from any or all of 3 places:

- The data input to the adapter
- A configuration file
- The adapter settings

All settings are merged with each other as they are encountered. The values prevailing at the last merge are used for any given invocation. The precedence is the ranking specified above (the opposite order in which these values are encountered). Configuration can be expressly dropped by a merge. This is can only be done for **<header>** tags. It is achieved by having a tag as follows:

<header name="SOAPAction"/>

The settings and their values are described in the following table:

Setting	Description	Default
mode	Dictates 'synchronousness' it can be: `synchronous`:- the caller will block expecting a response `startAsynchronousRequest` - identical in implementation to "`synchronous`"; however this is designed to match calls to "`waitForAsynchronousCompletion`" `waitForAsynchronousCompletion` - the caller will wait on Portrait async support providing a response. When this mode is selected the setting `transactionId` is required and `data/header` are ignored	synchronous
endPointURL	The target of the HTTP request	none
proxyServer	An optional name of a proxy server to use when making this call.	none
proxyUsername	An optional username to use with the above proxy server when making this call.	none
proxyPassword	An optional password to use with the above proxy server when making this call. Note that for the proxy username and password to be used, both settings must have a value.	none
timeout	The time in seconds (or the literal '`infinite`') before the transaction will be deemed to have failed	infinite
parseRequest	This indicates if the input data is to be passed out as the message in its entirety (false) or whether it is parsed for the keywords that may appear here (true) e.g. `mode`, `endPointURL`, `timeout` etc. It is best to avoid parsing the input data if the configuration can be adequately expressed via the adapter settings or configuration file.	true
header	These are headers that will be placed in the HTTP request. They are expressed as: `<header name="headername">headervalue</header>` and will create the HTTP header: headername: headervalue	none
data	The data to send to the endPointURL	null
registryKey	Describes a registry key containing a single string value "`ConfigurationFile`" that indicates a file path containing configuration in this XML format. This value is only ever used when interpreting the adapter configuration via adapter settings in the config suite.	none

transactionId	The id of the transaction that is to be looked up in Portrait's async support.	none
HTTPVersion	The version of HTTP to be used; this can be '1.0' or '1.1' Note: That the equivalent of Hotfix for Microsoft KB839942 for MSXML4 SP2 must be installed in order to use HTTP '1.1' without failure. Even with this Hotfix Timeouts for HTTP 1.1 must be less than 30 seconds ('infinite' does not fail but will timeout after 30 seconds)	1.1
clientCertificate	The name of the client certificate to send as part of this request. This name is the CN= value of the certificate's Subject field. The specified certificate is located in the certificate store of the calling machine.	none (no certificate is sent)

SSLCertificate	Specifies properties of the server SSL certificate. Through this parameter you can control which 'certificate errors' are ignored.
	The following children can be mixed and repeated within this tag. Their meaning is documented by Microsoft under the name SXH_OPTION_IGNORE_SERVER_SSL_CERT_ERROR_FLAGS.
	The flags are or'ed together - except the NoErrors flag which will unset any previously set flags. In the following examples 'n' can be a numeric expressed in hex (0x3300), octal (031400), or decimal (13056).
	`<ignore>n</ignore>`
	`<ignore>UnknownCA</ignore>`
	`<ignore>WrongUsage</ignore>`
	`<ignore>CNInValid</ignore>`
	`<ignore>DateInvalid</ignore>`
	`<ignore>AllErrors</ignore>`
	`<ignore>NoErrors</ignore>`
		Connections will be failed if the server SSL certificate has any of the suppressible errors

The following is an example of a configuration specification that may be used in the adapter settings (this is the only place where **<registryKey>** is appropriate):

```
<adapterConfig>
    <mode>synchronous</mode>
    <registryKey>SOFTWARE\PST\XMLOverHTTPsystem</registryKey>
</adapterConfig>
```

The following could be the contents of the file referred to by the above registry key:

```
<?xml version="1.0" encoding="UTF-8"?>
<adapterConfig>
    <endPointURL>http://localhost/XOH/Service1.asmx</endPointURL>
    <parseRequest>true</parseRequest>
    <timeout>100</timeout>
    <transactionId>none</transactionId>
    <header name="SOAPAction">
        http://tempuri.org/WebMethod
    </header>
    <header name="Content-Type">
        text/xml; charset=utf-8
    </header>
</adapterConfig>
```

Soap

The following is an example of calling a simple Soap web service method that takes a single input string **WebMethodInput**. Note that in this example the entire configuration is being specified in the input to the adapter

```
<request>
    <mode>synchronous</mode>
    <parseRequest>true</parseRequest>
    <endPointURL>
        http://localhost/XOH/Service1.asmx
    </endPointURL>
    <timeout/>
    <header name="SOAPAction">
        http://tempuri.org/WebMethod
    </header>
    <header name="Content-Type">
        text/xml; charset=utf-8
    </header>
    <transactionId/>
    <data>
        <soap:Envelope
        xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
        xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
        xmlns:xsd="http://www.w3.org/2001/XMLSchema">
            <soap:Body>
                <WebMethod xmlns="http://tempuri.org/">
                    <WebMethodInput>hello</WebMethodInput>
                </WebMethod>
            </soap:Body>
        </soap:Envelope>
    </data>
</request>
```

The response returned from the adapter would look like

```
<response>
    <status>
        <statusCode>200</statusCode>
        <statusText>OK</statusText>
    </status>
    <header name="Server">Microsoft-IIS/5.0</header>
    <header name="Date">Fri, 23 Apr 2004 15:10:00 GMT</header>
    <header name="X-Powered-By">ASP.NET</header>
    <header name="Connection">keep-alive</header>
    <header name="X-AspNet-Version">1.1.4322</header>
    <header name="Cache-Control">private</header>
    <header name="Content-Length">487</header>
    <data>
        <soap:Envelope
        xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema">
            <soap:Body>
                <WebMethodResponse xmlns="http://tempuri.org/">
                    <WebMethodResult>
                        WebMethod response
                    </WebMethodResult>
                </WebMethodResponse>
            </soap:Body>
        </soap:Envelope>
    </data>
</response>
```

Oneway

.Net Web service methods can be marks as 'Oneway'. This is achieved in C# as follows:

```
[WebMethod]
[SoapDocumentMethod(OneWay=true)]
public void WebMethod( string WebMethodInput )
{
```

Calling this method will result in an HTTP status 202 with no returned data as follows:

```
<response>
    <status>
        <statusCode>202</statusCode>
        <statusText>Accepted</statusText>
    </status>
    <header name="Server">Microsoft-IIS/5.0</header>
    <header name="Date">Fri, 23 Apr 2004 15:10:00 GMT</header>
    <header name="X-Powered-By">ASP.NET</header>
    <header name="Connection">keep-alive</header>
    <header name="X-AspNet-Version">1.1.4322</header>
    <header name="Cache-Control">private</header>
    <header name="Content-Length">0</header>
    <data></data>
</response>
```

HTTP status

In all events where an HTTP request returns a status the output from the adapter will contain the status tags. It is possible that there will be no data or headers. The adapter will not trigger the FAIL outcome even if the HTTP status is not 200 (OK). It is the responsibility of transformers to determines if the call actually has done any useful work.

FAIL outcome

A data object will be placed in the output in order to model certain failures. The FAIL outcome will need an output named **Error** which is a data object of category **DataAccessTransaction** and type **ErrorDetail**.

Timeouts (irrespective of mode) will generate this output

1.7 Adapter examples

1.7.1 SOAP adapter

Use of the SOAP adapter is deprecated as withdrawal of support for the SOAP toolkit has been announced by Microsoft.

The SOAP adapter enables Portrait Foundation to communicate with Web Services using SOAP protocol. The adapter is not Web Service-specific and uses a low-level SOAP interface to build the SOAP request message and read the SOAP response message. This means that a single adapter component is used to communicate with all Web Services (as opposed to building a proxy component for each Web Service). On the other hand, this means that the adapter needs to be supplied with some Web Service-specific information at runtime.

The Web Service-specific information is supplied from two sources:

- Adapter input – the SOAP adapter expects to receive an XML document (as a BSTR) that contains all the values to be encoded within the SOAP request.

Additionally this document contains information about the expected contents of the response from the host system.

- Adapter configuration – a combination of registry settings and configuration files provide information on how the adapter should encode the request, the URL of the Web Service, and whether the adapter should send a request or wait for and process a response.

SOAP toolkit versions

The current version of the adapter uses Microsoft SOAP Toolkit version 3.0 to serialise and deserialise the SOAP messages. However, if at run time version 3.0 of the toolkit cannot be found then the adapter will attempt to use version 2.0.

Adapter input

The input to the adapter must be a Portrait Foundation XML document which contains all the necessary information to build the SOAP request message. It also contains all the necessary information to interpret the SOAP response message

Figure 8 – SOAP adapter XML requesting the adapter to call Add operation on the Calculator Web Service.

```
<?xml version="1.0"?>
<SoapRequest operation="Add"
                action="http://tempuri.org/action/Calc.Add"
                namespace="http://tempuri.org/message/"
                xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <Inputs type="context">
        <A type="integer">12</A>
        <B type="integer">3</B>
    </Inputs>
    <Outputs type="context">
        <Result type="integer" xsi:nil="true"/>
    </Outputs>
</SoapRequest>
```

A valid SOAP request XML has following properties:

- The root element must be `<SoapRequest>`
- The root element must have the following attributes as defined in the WSDL file
  - `operation` – name attribute of the operation tag
  - `action` – value of `SoapAction` attribute of the `soap:operation` tag
  - `namespace` – value of `Namespace` attribute of the `soap:body` tag
- XSI namespace definition
- The root must contain a child `<Inputs>`
- The `<Inputs>` element must be a valid Data Context node as defined in Portrait XSD. The properties of a Data Context must have the same name and type as defined in the appropriate `<part>` tag in the WSDL file
- The root must contain a child `<Outputs>`
- The `<Outputs>` element must be a valid Data Context node as defined in Portrait XSD. The properties of a Data Context must have the same name and type as the operation outputs defined in a WSDL file. Value of an output should be set to NULL (`xsi:nil="true"`).

If we compare the SOAP XML above with the WSDL file example below we can see that the SOAP XML satisfies all the requirements.

Figure 9 –WSDL file of the Calculator Web Service.

```
<definitions >
```

```
    <types>
      <xs:schema targetNamespace="http://tempuri.org/type"/>
  </types>
  <message name="Calc.Add">
    <part name="A" type="xsd:double"/>
    <part name="B" type="xsd:double"/>
  </message>
  <message name="Calc.AddResponse">
    <part name="Result" type="xsd:double"/>
  </message>
  <portType name="CalcSoapPort">
    <operation name="Add" parameterOrder="A B">
      <input message="wsdlns:Calc.Add"/>
      <output message="wsdlns:Calc.AddResponse"/>
   </operation>
  </portType>
  <binding name="CalcSoapBinding" type="wsdlns:CalcSoapPort">
    <stk:binding preferredEncoding="UTF-8"/>
    <operation name="Add">
      <soap:operation soapAction="http://tempuri.org/action/Calc.Add"/>
    </operation>
  </binding>
  <service name="Calc">
    <port name="CalcSoapPort" binding="wsdlns:CalcSoapBinding">
      <soap:address location="http://loaclhost/Calc.wsdl"/>
    </port>
  </service>
</definitions>
```

Adapter output

The adapter output is a Portrait XML document representation of the data context defined by the **<Outputs>** tag of the adapter input (see above).

Figure 10 - The output from SOAP adapter after calling Add method of the Calculator Web Service.

```
<?xml version="1.0" encoding="UTF-8"?>
<SoapResult type="context" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <Result type="integer">15</Result>
</SoapResult>
```

The output data context has the same properties as defined in the input SOAP Adapter XML. The properties must have the same name as the output parameters that are defined in a WSDL file. The output data context does not have to contain all the output parameters of a Web Service.

Complex type support

The description of the adapter inputs, above, states that the **<Inputs>** and **<Outputs>** elements must be valid Data Context nodes. However, the current adapter only supports simple types within the data context, i.e. not Data Objects or Data Object Collections. Complex XML may be embedded in a single string within a data context (see below), but in order to send complex types in their own right, different versions of the **<Inputs>** and **<Outputs>** elements must be used with the type attribute set to "xml".

Figure 11 - Example of XSL transformation that constructs complex XML.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
        xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:output     method="xml"
                        version="1.0"
                        encoding="UTF-8"
                        indent="no"/>
    <xsl:template match="/">
        <SoapRequest    operation="SaveClient"
                            action="urn:ComplexTypes/SaveClient"
                            namespace="urn:ComplexTypes">
            <Inputs type="xml">
                <SaveClient xmlns="urn:ComplexTypes">
                    <Client>
                        <FirstName xmlns="">Jim</FirstName>
                        <Surname xmlns="">Clark</Surname>
                        <Address>
                            <Line1 xmlns="">The Smith Centre</Line1>
                            <Line2 xmlns="">The Fairmile</Line2>
                            <Town xmlns="">Henley-on-Thames</Town>
                            <County xmlns="">OXON</County>
                            <Postcode xmlns="">RG9 6AB</Postcode>
                        </Address>
                    </Client>
                </SaveClient>
            </Inputs>
            <Outputs type="xml">
                <SaveClientResult/>
            </Outputs>
        </SoapRequest>
    </xsl:template>
</xsl:stylesheet>
```

Note that when using Microsoft SOAP Toolkit Version 3.0, it is essential that namespaces are specified correctly since the underlying MSXML technologies are particularly strict in this respect.

Figure 12 - Example of XSL transformation that extracts a value from complex XML.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
            xmlns:xsl=http://www.w3.org/1999/XSL/Transform
            xmlns:ns="urn:ComplexTypes">
    <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="no"/>
    <xsl:template match="/SoapResult">
        <Outputs type="context">
            <Response type="string">
                <xsl:value-of
                    select="ns:SaveClientResult/ns:Client/FirstName"/>
            </Response>
        </Outputs>
    </xsl:template>
</xsl:stylesheet>
```

Embedded XML

It is very common for a Web Service to have an output parameter of type string that in runtime contains an XML document. It this case, the XML document will be encoded and become the value of a property in the output data context.

Figure 13 – Example of an XML document embedded within a data context XML.

```
<?xml version="1.0" encoding="UTF-8"?>
<SoapResult type="context" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <Result type="string">
        &lt;Collection&gt;
            &lt;Item&gt;Red&lt;/Item&gt;
            &lt;Item&gt;Green&lt;/Item&gt;
            &lt;Item&gt;Blue&lt;/Item&gt;
        &lt;/Collection&gt;
    </Result>
</SoapResult>
```

This is correct behaviour; however, very often we need to process the returned XML document further. What we need is a transformation that extracts and decodes the XML document and returns it as standalone rather than an embedded XML. The XML document can then be processed by the next output transformer.

Figure 14 – Example of XSL transformation that extracts the embedded XML.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>
    <xsl:template match="/SoapResult/Result">
        <MyXml><xsl:value-of select="." disable-output-escaping="yes"/></MyXml>
    </xsl:template>
</xsl:stylesheet>
```

Figure 15 – Result of the transformation.

```
<?xml version="1.0" encoding="UTF-8"?>
<MyXml>
    <Collection>
        <Item>Red</Item>
        <Item>Green</Item>
        <Item>Blue</Item>
    </Collection>
</MyXml>
```

SOAP headers

Web Services may require information to be placed in the Header part of the SOAP message. Typically this information is supplementary to the main request data passed in the message body e.g. a session or transaction id.

The SOAP adapter input may optionally contain **<InputHeaders>** and **<OutputHeaders>** tags to specify data that must be transmitted in the message headers. Note that the **<InputHeaders>** tag requires the name of the enclosing header to be specified.

Figure 16 - Adapter input including a SessionId request header element and a TransactionID response header element.

```
<?xml version="1.0"?>
<SoapRequest operation="Add"
             action="http://29empura.org/message/Add"
             namespace="http://29empura.org/message/"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <Inputs type="context">
        <A type="integer">12</A>
        <B type="integer">3</B>
    </Inputs>
    <InputHeaders type="context" name="RequestHeader">
        <SessionID type="string">1234</SessionID>
    </InputHeaders>
    <Outputs type="context">
        <Result type="integer" xsi:nil="true"/>
    </Outputs>
    <OutputHeaders type="context">
        <TransactionID type="string"/>
    </OutputHeaders>
</SoapRequest>
```

The example shown above will generate the following SOAP request:

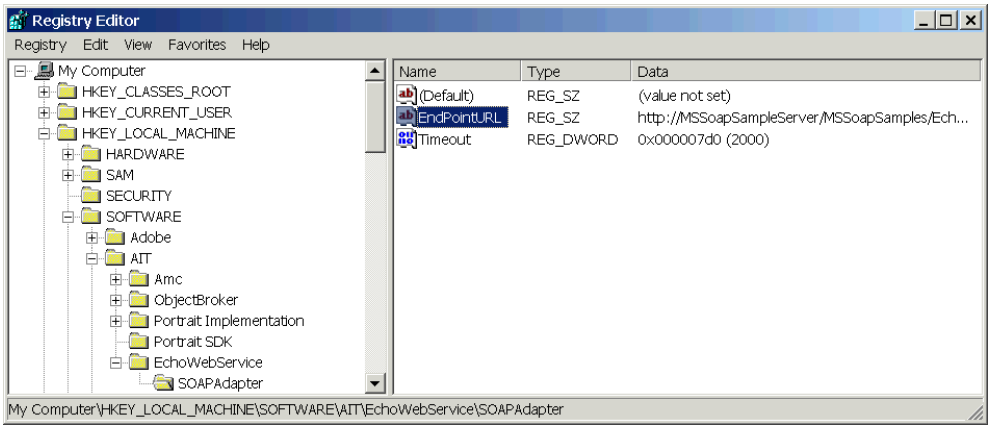
Figure 17 - Example SOAP request including SessionId request header element and a TransactionID response header element.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<SOAP-ENV:Envelope
    xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Header>
        <SOAPSDK1:RequestHeader
                        xmlns:SOAPSDK1="http://29empura.org/message/">
            <SOAPSDK1:SessionID>1234</SOAPSDK1:SessionID>
        </SOAPSDK1:RequestHeader>
    </SOAP-ENV:Header>
    <SOAP-ENV:Body>
        <SOAPSDK2:Add
                        xmlns:SOAPSDK2="http://29empura.org/message/">
            <SOAPSDK2:A>12</SOAPSDK2:A>
            <SOAPSDK2:B>3</SOAPSDK2:B>
        </SOAPSDK2:Add>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Basic configuration

The adapter configuration is stored in the registry within the HKEY_LOCAL_MACHINE section. The registry path must be configured in the properties dialog of the adapter in a system protocol stack. For example **SOFTWARE\PST\EchoWebService\SOAPAdapter** is valid registry path for the echo web service sample.

Figure 18 - SOAP adapter configuration stored in registry.



The **EndPointURL** property contains URL address of the Web Service and **Timeout** property contains maximum number of milliseconds that a Web Service request can take. The **SOAPFormatting** property indicates the binding style and body use values when constructing requests and reading responses. Valid values are "DocumentLiteral" and "RPCEncoded".

Advanced configuration

In addition to the configuration properties described above, an extra registry property is available, **ConfigurationFile**. If this property is present then **all** adapter configuration for the system concerned is read from the specified file. This file contains an XML document, an example of which is shown below:

Figure 19 - Example adapter configuration document with transaction-specific settings for the CalculatorAdd DAT.

```
<?xml version="1.0" encoding="UTF-8"?>
<AdapterConfig>
    <Service    EndPointURL="http://MSSoapSamples/Calc/Service/Rpc/IsapiVb/Calc.wsdl"
                Timeout="2000"
                SOAPFormatting="RpcEncoded"/>
    <Mode       WaitInbound="0"
                ResponseRequired="0" />
    <CalculatorAdd>
        <Service    EndPointURL="http://DotNetCalcServer/Calc/Calc.asmx"
                    Timeout="2000"
                    SOAPFormatting="DocumentLiteral"/>
        <Mode       WaitInbound="0"
                    ResponseRequired="0" />
    </CalculatorAdd>
    <TxnResponse>
        <Service    Timeout="20000"
                    SOAPFormatting="DocumentLiteral"/>
        <Mode       WaitInbound="1"
                    ResponseRequired="0" />
        <Transaction
                    Origin="None"
                    Name="TransactionID"/>
    </TxnResponse>
</AdapterConfig>
```

A valid adapter configuration XML document has following properties:

- The **root** element must be **<AdapterConfig>**

- The **root** element may contain one or more DAT-specific child elements where each DAT-specific child element's tag must match a DAT system name.
- Each DAT-specific child element may contain a child **<Mode>**.
- The Mode element must have following attributes
  - **WaitInbound** – Set to "0" to send a SOAP request, set to "1" to wait for an inbound SOAP request. If this attribute is set to "1", the adapter will expect to receive the whole SOAP message, not just the message body.
  - **ResponseRequired** – for future use (optional)
- Each DAT-specific child element must contain a child **<Service>**.
- The **Service** element must have following attributes
  - **EndPointURL** – URL of the Web Service (required if the Mode element is absent or WaitInbound is set to "0")
  - **Timeout** – maximum duration of the request in milliseconds
  - **SOAPFormatting** – specifies the binding style and body use. This attribute is optional and defaults to "RPCEncoded".
- Each DAT-specific child element may contain a child **<Transaction>**. This element is required under the following circumstances:
  - A unique transaction id (a GUID) must be generated by the adapter for inclusion in the SOAP request to the host system. (Note that if a non-zero length transaction id string is passed into the adapter then a new id will NOT be generated).
  - The adapter is expected to wait for the arrival of a SOAP request that contains a unique transaction id. (The Mode element's WaitInbound attribute is set to "1")
- The **Transaction** element must have following attributes
  - **Name** – the tag name of the **<Inputs>** (or **<InputHeaders>**) or **<Outputs>** (or **<OutputHeaders>**) child element that represents the transaction id.
  - **Origin** – Set to "Request" to generate and send a unique transaction id in the SOAP request, set to "None" to wait for an inbound SOAP request containing a unique transaction id.
- The root element must contain a child **<Service>** element (defined above) if the system contains DATs that are not represented by a DAT-specific child element. The attributes of the root **<Service>** element will apply to all DATs that are not represented by a DAT-specific child element.
- The root element must contain a child **<Mode>** element (defined above) if the system contains DATs that are not represented by a DAT-specific child element. The attributes of the root **<Mode>** element will apply to all DATs that are not represented by a DAT-specific child element.

Implementation

The adapter uses **IsoapConnector**, **IsoapSerializer**, and **IsoapReader** interfaces to create a low level SOAP request and read the results. The adapter uses MSXML4 to parse the input XML that contains all the information needed to make a low level SOAP request. In addition, **PortraitData2Xml** and **Xml2PortraitData** helper classes are used to read/create input/output data context.

The adapter uses the synchroniser component to wait for inbound SOAP requests (no responses are currently produced). A service to receive the requests must be implemented as part of the project's solution.

1.7.2 MSMQ adapter

The MSMQ adapter implementation shows how to use the framework to integrate to a host system using Microsoft Message Queue. In this fully functional example, we show how to use two queues to send and receive messages as well as to achieve asynchronous communication.

The example is in fact an implementation of three adapters – outbound, inbound and duplex.

MSMQ Outbound

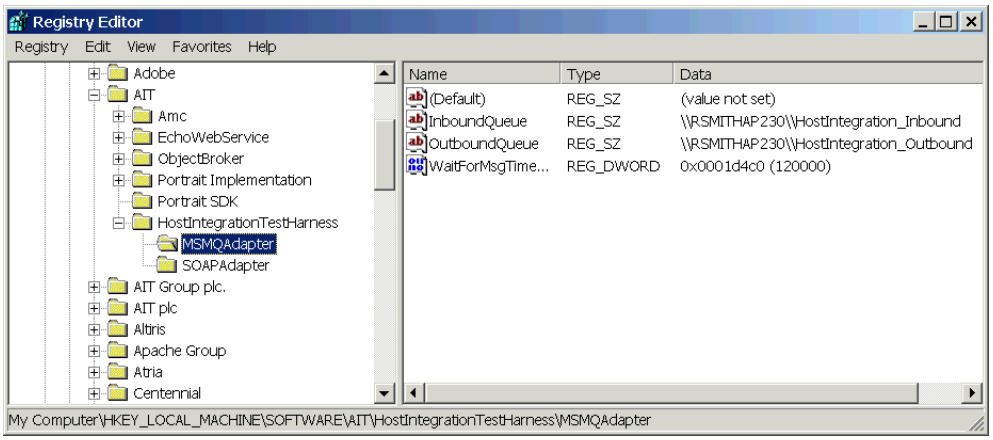
The adapter implements the outbound part of asynchronous communication with MSMQ.

The adapter sends input data to the specified queue and returns a data context which contains the correlation message id.

Initialisation

The implementation of the **Initialise()** method reads the outbound queue setting from the registry. The outbound queue settings are stored in **m_strQueuePathName**.

Figure 20 - MSMQ adapter configuration stored in registry



Processing

The implementation of the **ProcessData()** method does the following

- Creates the required MSMQ components
- Opens the queue specified in **m_strQueuePathName**
- Sets the message body to be the value of **vInputData**
- Sends the message
- Reads the id of the message
- Creates a new data context object with a single string property called CORRELATION_ID
- Sets the property value to be the id of the message
- Outputs the new data context as **pvOutputData** and returns

Data format requirements

The MSMQ API accepts VARIANTs which means that the adapter is capable of passing any input data into MSMQ. However, MSMQ is only capable of handling simple types or a persistent COM object that supports Idispatch and Ipersist.

The output data is VARIANT of type VT_BSTR.

MSMQ Inbound

The adapter implements the inbound part of asynchronous communication with MSMQ.

Input to the adapter is a data context which contains the correlation message id returned from the outbound adapter. The adapter searches the specified inbound queue for a correct message and returns the message body as its output. When the requested message is not found the adapter will wait for the specified amount of time.

Initialisation

The implementation of the **Initialise()** method reads the adapter setting from the registry. The outbound queue setting is stored in **m_strQueuePathName** and the time out setting is stored in **m_vWaitForMsg**.

Processing

The implementation of the **ProcessData()** method does the following

- Reads the CORRELATION_ID property of the input data object
- Creates the required MSMQ components
- Opens the inbound queue
- In the loop peeks messages looking for a message with the specified CORRELATION_ID
- Once the correct message is found, the adapter removes it from the queue
- The message body is returned as the value of **pvOutputData**

Data format requirements

The input data is VARIANT of type VT_BSTR.

The output data is VARIANT of the same type as the type of the MSMQ message body.

MSMQ Duplex

The adapter combines the functionality of Inbound and Outbound MSMQ adapters to deliver synchronous MSMQ communication. From the framework point of view, the adapter returns output data directly as opposed to the intermediate CORRELATION_ID in the case of asynchronous adapters.