# Preamble

Before someone decides to build grid services they should at least be familiar with basic grid service architecture and also that of statefull grid services if they plan to create asynchronous services and/or statefull services. To read more about grid service architecture and grid middleware to refresh your memory or get basic understanding please read some of the following basic information:  
[Overview Papers on Grid Computing](http://www.globus.org/alliance/publications/papers.php#Overview%20Papers)  
[Globus Best Practices](http://globus.org/toolkit/docs/4.0/best_practices.html)

## Overview

|  |
| --- |
| [https://wiki.cagrid.org/download/attachments/2917693/worddav25dabc26311c17cb912d6897a3516f1c.png[https://wiki.cagrid.org/images/icons/link_attachment_7.gif](https://wiki.cagrid.org/download/attachments/2917693/worddav25dabc26311c17cb912d6897a3516f1c.png?version=1)](https://wiki.cagrid.org/download/attachments/2917693/worddav25dabc26311c17cb912d6897a3516f1c.png?version=1) |
| **Introduce Overall Service Creation Process** |

The Introduce toolkit is designed to support the three main steps of service development: 1) Creation of Basic Service Structure. The service developer describes at the highest level some basic attributes about the service such as service name and service namespace. Once the user has set these basic service configuration properties, Introduce will create the basic service implementation, to which the developer can then add application-specific methods and security options through the service modification steps. 2) Service Modification. The modification step allows the developer to add, remove, and modify service methods, properties, resources, service contexts, and service/method level security. In this step, the developer can create a strongly-typed service interface using well-defined, published schemas, which are registered in a system like the Mobius GME, as the type definitions of the input and output parameters of the service methods. One the operations are added to the service the developer will then be able to add the logic which implements the methods. 3) Deployment. The developer can deploy the service which has been created with Introduce to a Grid service container (e.g., a Globus, Tomcat, or JBoss service container). A service developer can access the functions required to execute these three steps through the Graphical Development Environment (GDE) of Introduce. The runtime support behind the GDE functionality is provided by the Introduce engine, which consists of the Service Creator, Service Synchronizer, and Service Deployer components. The toolkit provides an extension framework that allows Introduce to be customized and extended for custom service types and discovery of custom data types. In the following sections, we describe the software prerequisites, the Introduce Graphical Development Environment, the Introduce Engine, and the Introduce Extension Framework in greater detail.

### Software Prerequisites

* Java 1.5 or greater ([http://www.java.sun.com[https://wiki.cagrid.org/images/icons/linkext7.gif](http://www.java.sun.com/)http://www.java.sun.com[https://wiki.cagrid.org/images/icons/linkext7.gif](http://www.java.sun.com/)](http://www.java.sun.com))
* Apache Ant 1.6.5 or 1.7.0 ([http://www.ant.apache.org[https://wiki.cagrid.org/images/icons/linkext7.gif](http://www.ant.apache.org/)http://www.ant.apache.org[https://wiki.cagrid.org/images/icons/linkext7.gif](http://www.ant.apache.org/)](http://www.ant.apache.org))
* The Globus Toolkit Version 4.0.x ([http://www.globus.org[https://wiki.cagrid.org/images/icons/linkext7.gif](http://www.globus.org/)http://www.globus.org[https://wiki.cagrid.org/images/icons/linkext7.gif](http://www.globus.org/)](http://www.globus.org))

**Changes from Introduce 1.2**

* Fixed circular schema bug in filesystem data type discovery tool
* Fixed bug in updater that was looking in the wrong place for introduce.extension.properties and template when trying to set the patch version
* Added new build-deploy.xml to fix problems with undeploy (upgrader needs to handle the copy in)
* New heuristic for schema imports
  + if introduce imports the schema it will only set namespace excludes or type maps for that schema
  + if it is desired to do this for the schemas that schema imports than those schemas must be added and handled appropriately individually. This is different than in prior versions.
  + Prior versions of introduce would simple namespace exclude any namespace that were imported from a schema that was set to generate stubs false. However this is not always the desired behavior
* Added button to enable user to control whether or not stubs will be generated for imported schema.
* Added to introduce model so that services and methods can have extension data added to them
* Added an introduce log viewer component to the GUI.
* Added new extension type called INTRODUCE\_GDE which enables adding grape components to the UI through extensions
* Added another subscribe method to base clients that takes in a NotificationCallback so that clients do have to overload the deliver method if they want to implement a notification callback on there own or use another.
* Complete refactor of authorization within introduce. Introduce now has a Authorization Extension that enables authorization plugins to the Introduce Generated PDP. This is much cleaner and removes the hard coded gridgrouper and csm support and now opens the doors to other authorization plugins to be generated rapidly.
* Added ability for introduce generated clients to have a setAnonymousPrefered. This will enable controlling whether or not the client will attempt to connect anonymously or not. The default is set to true and can now be changed programmatically.
* Added to introduce service extension so that service extension providers can plug a deployment validator into the service that will be executed before the service is deployed. This will enable service extension providers to validate the service is properly configured prior to deployment.

**Known Issues**

* Introduce cannot generated services in a directory which has any parent directory that has a space in it.
* Introduce creates duplicate wsdl imports in the wsdl documents which may cause problems with some BPEL orchestration tools.

**Introduce Graphical Development Environment**

|  |
| --- |
| [https://wiki.cagrid.org/download/attachments/2917693/introduceGDE.png[https://wiki.cagrid.org/images/icons/link_attachment_7.gif](https://wiki.cagrid.org/download/attachments/2917693/introduceGDE.png?version=1)](https://wiki.cagrid.org/download/attachments/2917693/introduceGDE.png?version=1) |
| **Introduce Graphical Development Environment (GDE)** |

The Introduce Graphical Development Environment (GDE) can be used to create, modify, and deploy a grid service. It is designed to be very simple to use, enable using community accepted data types, and provide easy configuration of service metadata, operations, resources, and security. It also allows customized plug-ins to be added for such things as discovering data types from grid repositories and for creating custom service style design templates.  
The Introduce GDE contains several screens and options for the service developer to 1) create a new service, 2) modify an existing service, 3) discover and use published data types in order to create strongly-typed service methods, and 4) deploy the service.

**Launching Introduce**

Introduce can be lauched from the command line via ant. You will simply need to open a terminal and change directory to the cagrid installation directory execute:

https://wiki.cagrid.org/download/attachments/2523281/Tutorial-Utilities-Terminal.png

ant introduce

**Service Creation**

|  |
| --- |
| [[https://wiki.cagrid.org/download/attachments/2917693/createScreen.png](https://wiki.cagrid.org/download/attachments/2917693/createScreen.png?version=1)[https://wiki.cagrid.org/images/icons/link_attachment_7.gif](https://wiki.cagrid.org/download/attachments/2917693/createScreen.png?version=1)](https://wiki.cagrid.org/download/attachments/2917693/createScreen.png?version=1) |
| **Introduce GDE Service Creation Component** |

The service creation component, shown below, enables the developer to create a new grid service. Using the creation interface, the service developer can provide basic information about the service such as:  
**Creation Directory**  
*The creation directory is the location of which the grid service will be generated.***Service Name**  
*Service name is the name that will be used to generate the service. The service name must be a valid java identifier.***Package Name**  
*The package name is the base package to be used when generated the grid service source code.***Namespace**  
*The namespace is the namespace to be used when defining the WSDL of the service.*

The developer also has the ability to add service extensions. A service extension is an Introduce plug-in (see [Extensions](https://wiki.cagrid.org/display/introduce13/Extensions)), which is designed to add customizations to the service. For example, service extensions might add pre-defined operations, resources/resource properties, or security settings. They enable the development of custom service types with predefined methods, which must be implemented. They also enable Introduce to run the custom code implemented in the plug-in, which makes modifications to the underlying service being created. This capability allows the specialization of Introduce to support domain specific common scenarios, further abstracting the individual service developer from responsibilities related to the deployment of grid technologies in a production environment. Once the information has been entered and extensions, if any, have been selected, the user will select the create button. Once the creation button is selected the Introduce creation engine will begin generating the service. After the service is generated is will be compiled and the Modification component will be displayed. For a list of available caGrid Service extension and their documentation please look [here](https://wiki.cagrid.org/display/introduce13/Extensions).

**Service Modification**

Service modification can be performed on any new or previously modified Introduce generated service. The service developer can perform a series of operations in order to begin to customize the grid service or modify the existing grid service. The overall flow in the modification of a grid service is to first use the namespaces tab to be sure that all the data types that are desired to be used in the grid service have been selected and added to the service. Next the service can choose to either add/remove or modify operations, metadata in the form of resource properties, service properties, security setting, and service contexts. The following sections will describe in detail how each of the components of the modification viewer can be used to modify the grid service to achieve desired functionality. By selecting the "Modify Service" button on the main menu a prompt will apear to enable choosing the service to be modified. Once the desired directory containing the service to be modified is selected the modification viewer component will be launched. The modification viewer contains 6 main areas where modifications can occur on the main service:

**Data Types**

|  |
| --- |
| [https://wiki.cagrid.org/download/attachments/2917693/importTypeScreen.png[https://wiki.cagrid.org/images/icons/link_attachment_7.gif](https://wiki.cagrid.org/download/attachments/2917693/importTypeScreen.png?version=1)](https://wiki.cagrid.org/download/attachments/2917693/importTypeScreen.png?version=1) |
| **Introduce GDE Service Modification Component** |

The first task in the modification of a grid service is to discover the data types that are desired to be used as the input and output types of methods of the service and the data types for describing the resource properties of the service. Adding a data type to the service is equivalent to copying schemas into the schema location of the service and importing the schemas into the WSDL file so that the types can be referenced by the service. This is done via the "Types" tab of the Graphical Service Modification Environment. This tab shows the current types the service is using, and provides access to the data type discovery components (such as the Mobius GME), for selecting and configuring additional types. The "Select Type" frame enables several types of ways to locate data types and bring them into the service. Currently there are two main discovery mechanisms (Globus, and File System) that come with introduce, however, this is extensible via the Discovery Extension described in the Extensions section. For a link of available Data Type Discovery Extensions and their documentation please look [here](https://wiki.cagrid.org/display/introduce13/Extensions). Once a set of data types from a namespace are brought into the service the user has the ability to describe how these data types will be mapped into there respective Java classes. This can, by default, be done automatically by Introduce via Axis. By default, Axis will create new java beans for each data type and also provide a serializer and deserializer for those objects. If for example, a set of objects already exist for this particular data types then a user can decide to provide there own classes and serialization/deserialization factories.

**Importing Data Types**

Using the Introduce GDE, developers can obtain data types that they want to use for operation input parameters and return types from any data type discovery plug-in. Utilizing common and standard data types, which are defined outside of any application-specific service, enables the creation of strongly typed grid service interfaces. This increases service-to-service interoperability. Once a data type is chosen through the GDE, the data type definition is retrieved, written into the *schema/<service\_name>* location of the service, and imported for use in the service WSDL description so that Java beans can be generated and the data types can be programmatically used.  
The Introduce toolkit comes with a set of pre-installed discovery plug-ins, such as the basic file system browser, which can be used to locate and import schemas.  
When importing a data type there are several options in where to acquire the data type definitions. Introduce is build in with the following data type definition tools, however, this is a pluggable area of Introduce. Data Type Discovery Extension can be found [here](https://wiki.cagrid.org/display/introduce13/Extensions). Once a data type is imported using an import tool that data type can be customized for the generation of Java Beans. If you select data type on the left you will see in the lower left panel that the namespace and package name have been listing. This is called the namespace to package map. This will determine the package name of the Java Beans that get created for the data type. Feel free to alter the package name if the Introduce suggested package name does not work well.

* ***File System Data Type Importing***  
  The File System tab of the Import Data Types Panel enables the developer to load in schemata which contain data types they wish to use the service from the local filesystem. The developer can browse to choose the schema they wish to import, and the click the add button. Once the add button is clicked the schema and any locally included or imported schemata will be copied to the services schema location in the *schema/<service name>* directory.
* ***Globus Data Types Importing***  
  The Globus Data Types extension enable the developer to import schema from the Globus toolkit into their service. There is a drop down containing a list of the available schemata from the current installation of the Globus Tooolkit. Once a namespace has been selected and the add button has been selected then the schema will be added to the service's available data types list.

**Re-Importing a Modified Data Type**

Introduce will enable re-importing of a data type if the developer wants to re-import a particular schema which may have been modified or extended. In order to do this you must make sure the *"Namespace Type Replacement Policy"* configuration setting in the Introduce *"Configuration-->Preferences"* menu is set to *"warn"*. Once this is done you can simply browse back to the data model and import it again.

**Using Custom or Pre-Existing Java Beans**

Once a namespace and corresponding data types have been imported into the service each data type can be further customized. For a particular data type one can chose to use a Custom Java Bean that already exists instead of having Introduce create the java beans for the service. This can be accomplished by selecting the ***Configure Types*** tab in the Types tab and then the ***+*** button beside the *Customize Bean* label. This will drop down the customization panel for that particular data type. In this panel, to support using a custom bean for the selected data type definition, the developer must fill out the three fields: the classname of the bean to be used (make sure the package name above matches the package name being entered for the custom beans classname), the deserializer factory class, and the serializer factory class. For more information on using custom serialization or what it means to be a custom bean please refer to the most recent Globus documentation on *type mapping*.

**Services**

The services tab of the GDE is the main tab for editing the service and the service contexts associated with this service. It contains a tree which shows all the services, or service contexts, which are part of your service, their methods, and their resource properties. This tree is the main view of the services which will be deployed as part of a deployment of the introduce service. The developer has the ability to add new service contexts, add/remove/modify operations on a particular service, and add/remove/modify resource properties of a particular service.  A service context is just another service that is being built with this service.  The services are deployed together as a bundle and share the same source tree and libraries.  This makes it easy to compose services which utilize each other such as in utilizing WSRF with a factory service and a service used to manipulate the resources the factory can create.

|  |
| --- |
| [https://wiki.cagrid.org/download/attachments/2917693/serviceContextsScreen.png[https://wiki.cagrid.org/images/icons/link_attachment_7.gif](https://wiki.cagrid.org/download/attachments/2917693/serviceContextsScreen.png?version=1)](https://wiki.cagrid.org/download/attachments/2917693/serviceContextsScreen.png?version=1) |
| **Introduce GDE Services View** |

**Edit Service**

Once in the tree view of the Services tab you can right click on any service and select the Edit button. This will pop up a window, like the image below, that will enable the user to configure the service context. From this pop up window the resource framework options can be added or removed, service level security can be configured, and a description can be added describing the service.

**Resource Framework Options**

The resource framework options for a service will add or remove a particular type of functionality to the service resource. The supported resource framework options for this release of introduce are as follows

* **Custom**  
  Enables the user to provide thier own implantation of the resource class.
* **Singleton**  
  The service will only have one instance of the resource.
* **Lifetime**  
  The resources created by this service will support the WS-Lifetime specification and therefore the service will implement the setTerminationTime and destory operations as part of its wsdl.
* **Persistent**  
  The resources created by this service will automatically persist themselves including resource properties and registered notification consumers to the file system so that if the container is restarted or inadvertently dies they will come back to life once the container is up and running again.
* **Secure**  
  The resources will implement SecureResource and therefore have the getServiceSecurityProvider so that they can provide a security descriptor for each particular instance of the resource.
* **Notification**  
  The resources created by this service will automatically support the WS-Notification specification and therefore the service will implement the subscribe operation in its wsdl and the client will have operations to make subscriptions and utilize notifications.
* **Resource Property Access**  
  The resources will implement the getResourcePropterty, getResourceProperties and QueryResourceProperty operations and these methods will be exposed through the service's wsdl.

|  |
| --- |
| [https://wiki.cagrid.org/download/attachments/2917693/modifyServiceScreen.png[https://wiki.cagrid.org/images/icons/link_attachment_7.gif](https://wiki.cagrid.org/download/attachments/2917693/modifyServiceScreen.png?version=1)](https://wiki.cagrid.org/download/attachments/2917693/modifyServiceScreen.png?version=1) |
| **Introduce GDE Modify Service View** |

**Security**

**Authentication**  
Introduce exposes the functionality of Globus GSI through a set of panels which enable the user to customize security for the entire service or specific methods on an service context. The user can choose any of the GSI configuration scenarios such as Transport Level Security with Integrity and Secure Communication with Privacy. For detailed knowledge of what the configuration options for Secure Conversation or Secure Credentials are please refer to documentation for the [GSI[https://wiki.cagrid.org/images/icons/linkext7.gif](http://www.globus.org/toolkit/docs/4.0/security/)](http://www.globus.org/toolkit/docs/4.0/security/) framework.  
**Authorization**  
Introduce also enables configuring a particular service, operation, or resource, for authorization. [Authorization Extensions](https://wiki.cagrid.org/display/introduce13/Extensions) can be installed into introduce to enable graphical configuration of service or method level authorization. Graphical panels will enable the user to describe an authorization policy which must be met in order to give access to the particular service or operation.

**Operations**

The developer can add, remove, or modify operations on the service. To add an operation click on the Services tab as descibed in the ealier section, and then select the service in the tree that you wish to add the operation to. Once you have selected the service you can then select the **Add Method** button from the panel on the right. For modifying or removing an operation just select the operation from the Services tree on the left and select the **Modify** or **Remove** method button from the panel on the right.

**Creating or Modifying an Method**

|  |
| --- |
| [https://wiki.cagrid.org/download/attachments/2917693/createMethodScreen.png[https://wiki.cagrid.org/images/icons/link_attachment_7.gif](https://wiki.cagrid.org/download/attachments/2917693/createMethodScreen.png?version=1)](https://wiki.cagrid.org/download/attachments/2917693/createMethodScreen.png?version=1) |
| **Introduce GDE Method Modification Component** |

Each operation needs to have unique name. All new methods will be defaults the a name called "newMethod". This should be changed to the desired operation name and also a description, if desired, can be added to this operation. For each operation, the developer needs to set the input parameters, return type, and any fault types that can be thrown from each service method. The tabs within the **Modify Method** panel will allow you to set the Input, Output, Faults, and Security settings of the operation.

* **Input**  
  To set the input parameters of the operation first select the **Input** tab of the **Method Modification** panel. This will display a table of the input parameters for this method. The input data types can be selected from the types tree on the left. This tree represents the available data types which can be used by this service. To add a data type to the input parameters simply select the data type from the tree on the left by clicking it. Then either double click the item to add it to the input parameters table or use the Add button. If any input parameter is to be used as an array the array checkbox must be checked in the table on the right once the data type has been added to the table. Also, once an input parameter is added the name of the parameter is defaulted. This name can be edited by the developer by selected the cell in the name column and editing the text.
* **Output**  
  To set the return type of the operation first select the **Output** tab of the **Method Modification** panel. This will display the return type or this method. The output data types can be selected from the types tree on the left. This tree represents the available data types which can be used by this service. To set the return type simply select the data type from the tree on the left by double clicking it. If you would like to reset the operation back to have no return type, the default, you can click the "Clear Output Type" button. If any output type to be used as an array the array checkbox must be checked in the table on the right once the data type has been added to the table.
* **Faults** There are three ways to add faults, either choose a type from the types tree on the left which extends WSRF BaseFaultType and click the **Add From Type** button, select from a fault that already exists in the service that is being used somewhere else and reuse it from the **Used Faults** drop down and then click the **Add Used Fault** button, or create a new fault which will tell Introduce to create you a new fault type which extends the BaseFaultType by typing in the fault name in the **Fault Type Name** text box and then click the **Add New Fault** button. Adding faults enables you to throw back failure information back to the client that they can plan for. This enables your operations to be more user friendly with respect to known errors that you service operation might run into.
* **Security**  
  Clicking on the **Security** tab of the **Method Modification** panel enables the service developer to configure security settings for the particular operation. An individual operation can require a different level of authentication and authorization than other methods enabling the mixing of public and private methods within a published service. Introduce leverages the Globus GSI framework for Authentication and encryption and the Globus PDP framework as well as a custom Configurable Introduce PDP for authorization.
  + **Authentication**  
    Introduce exposes the functionality of Globus GSI through a set of panels which enable the user to customize security for the entire service or specific methods on an service context. The user can choose any of the GSI configuration scenarios such as Transport Level Security with Integrity and Secure Communication with Privacy. For detailed knowledge of what the configuration options for Secure Conversation or Secure Credentials are please refer to documentation for the [GSI[https://wiki.cagrid.org/images/icons/linkext7.gif](http://www.globus.org/toolkit/docs/4.0/security/)](http://www.globus.org/toolkit/docs/4.0/security/) framework.
  + **Authorization**  
    Introduce also enables configuring a particular service, operation, or resource, for authorization. [Authorization Extensions](https://wiki.cagrid.org/display/introduce13/Extensions) can be installed into introduce to enable graphical configuration of service or method level authorization. Graphical panels will enable the user to describe an authorization policy which must be met in order to give access to the particular service or operation.

**Using a Pre-Existing Operation Implementation**

The implementation of a described operation may already exist in another class which is provided by a jar file. You can tell Introduce not to stub this methods server side implementation but instead call a provided method implementation directly in the class provided. In order to use this functionality the **Provided** checkbox must be selected and the class name attribute must be filled out in the **Provider** tab. The class name attribute will point to the fully qualified class name of the class which implements this WSDL described operation. The jar file that contains the provided Class which implements this operation must also me copied into the lib directory of the service. This will ensure that the operation will be located at the time the operation is called on the service. For more information on this particular topic refer to the Globus Documentation on Operation Providers.

**Importing Operations**

Operations can also be imported from other services. Importing an operation enables the service to implement the exact same operation signature as described in the other service. This enables the service to have an operation which has the exact same WSDL signature of the operation which is being imported. This would enable either client to invoke this operation on either service. Importing can be done in two ways: (1) from an Introduce generated service, or (2) from a WSDL file. For case 1, importing from an Introduce service, the developer would browse and select the Introduce generated service which contains the operation to be imported. Once the Introduce service is selected a list of services which contain this method will be available to select from. Select the service from which you want to import the operation. The methods signature will be imported and the developer will be prompted to make sure to copy over the WSDL and XSD files needed to import the method into the *schema/<service name>* directory of the service. For case 2, if a method is described in another WSDL but the developer wants to implement this exact method from this WSDL. The developer must have the WSDL and corresponding XSD's in the *schema/<service name>* directory of the service. Then the developer will be able to browse those WSDL files and select the port type they wish to import the operation from. The importing of a method across services will assure not only that each service has completely protocol compatible methods but also that each service's method can be invoked by the same base client. This enables the notion of basic inheritance in grid services and is discussed further in the Introduce technical guide.

**Resource Properties**

Service state information and metadata in the form of resource properties can be added, removed and configured via the **Metadata** tab of the GDE Service Modification interface. The metadata elements which are added to the service can be populated by a file statically or managed dynamically within the service. Also, these metadata entities can be registered with an index service so that users can use the metadata to locate the service. Once the "Metadata" tab is clicked the left panel will contain a list of available data types that can be used for metadata and the right will contain the list of currently chosen data types. By double-clicking on a data type in the left panel it will be added to the main service's metadata list. Any of the service's metadata can be initially populated from a file if desired. If this is chosen then once the service is started up in the container the file will be used to populate the particular metadata object in the service. Each metadata in the service can also be selected to be published to an index service. This will enable some or all of the metadata to be used in to locate the service via an index service.

**Service Contexts**

A power user feature which can be enabled at modification time is the addition or removal of service contexts. A service context a sub-service or complimentary service which is used with the main service or some other service context. The service context is comprised of the service, resource, operations, and resource properties. So, in a sense, service context is exactly the same thing as the main service, except that it is not a singleton based resource and instances can be more dynamically created and or destroyed. Contexts can be added via the "Service Contexts" tab of the GDE Service Modification interface. Service contexts define additional of operations needed to support the desired service functionality. This is enabled by using WSRF capabilities of the Globus Toolkit. As an example, if an operation on the main service enables the user to query a database, that operation might create a resource in another context and return the handle to that context to the user as opposed to the full query result set. This secondary context can then enable the user to iterate through the query results. This is accomplished by operations or resource properties to this secondary service context which will be responsible for iteratively giving results to the user. It should be noted that multiple instances of these contexts can be created and executed concurrently; one for each query that comes in, for example. This style of grid service is supported by the WSRF specifications. Though the details of the WSRF-implementation of these concepts are abstracted away from developers its worth noting how they are realized, and this is described in detail in other sections. Introduce makes it easier for service developers to create such complex services, via the GDE, without having to fully understand the underlying service implementations. Anything that can be done to the main service, except service properties which are globally accessible can be added to a service context. For example, resource properties can be added and used to maintain state or for publishing metadata to an index service. Also, operations can be added to the service context and can also be implemented in the service itself or in the service's resource if they are acting on the state of the instance of the resource.  
A Stateful Grid service is comprised of several key components which make it able to maintain state and enable a client to invoke the service several time under the same context. A stateful grid service is composed of the service, a resource home, and the resource type. This service organization can be used in many different scenarios. For, example, when an operation on the service is invoked the service can be implemented to handle that operation, or if the operation is addressing a particular resource instance in the service, the service can lookup the resource and call whatever might be necessary to call on the particular addressed resource.

**Service Properties**

|  |
| --- |
| [https://wiki.cagrid.org/download/attachments/2917693/servicePropertiesScreen.png[https://wiki.cagrid.org/images/icons/link_attachment_7.gif](https://wiki.cagrid.org/download/attachments/2917693/servicePropertiesScreen.png?version=1)](https://wiki.cagrid.org/download/attachments/2917693/servicePropertiesScreen.png?version=1) |
| **Introduce GDE Service Properties View** |

Service properties are key value pairs which can be set at deployment time and are available to the server side implementation of the service at run time. This enables passing in configuration variables to the server side of the service at deployment. These key value pair properties can be declared in "Service Properties" tab of the GDE Service Modification interface. Once the "Service Properties" tab is clicked the main panel will show a table of the service properties. The bottom panel has an entry for which can be used to create a new service property. The properties will be confirmed and/or can be changed from there default values at service deployment time. The variables can then be accessed inside the user's implementation of the operations through the services *ServiceConfiguration* class. For example, if you add a property called *foo* under the service properties tab, and then save the service. Then go look at the source code for the <service package>.service.ServiceConfiguration.java class you will see that it now has available methods for *string getFoo()* and void *setFoo(string foo)*. These operations are now available to your service and can be used to pass properties into your service at deployment time as well as other users for configuring and sharing properties in your service. The <service package>.service.ServiceConfiguration.java contains a static method for obtaining an instance of itself called *getConfiguration()*. Any call to that operation from anywhere in the service will return the handle to the "ServiceConfiguration" instance and hence provide access to the service properties.

**Service Extensions**

|  |
| --- |
| [https://wiki.cagrid.org/download/attachments/2917693/extensionsScreen.png[https://wiki.cagrid.org/images/icons/link_attachment_7.gif](https://wiki.cagrid.org/download/attachments/2917693/extensionsScreen.png?version=1)](https://wiki.cagrid.org/download/attachments/2917693/extensionsScreen.png?version=1) |
| **Introduce GDE Extensions View** |

Services can select extensions to add to their Introduce service. These extensions can provide special functionality to the service that might be useful. For example, the caGrid Transfer extension enables the service developer to utilize the caGrid Transfer API in there service and client which can enable them to utilize xml based movement of data between service and client and vice-versa. For more information on available extensions that can be added to Introduce or its services please look at the [Extensions](https://wiki.cagrid.org/display/introduce13/Extensions) page.

**Deployment**

|  |
| --- |
| [https://wiki.cagrid.org/download/attachments/2917693/deployServiceScreen.png[https://wiki.cagrid.org/images/icons/link_attachment_7.gif](https://wiki.cagrid.org/download/attachments/2917693/deployServiceScreen.png?version=1)](https://wiki.cagrid.org/download/attachments/2917693/deployServiceScreen.png?version=1) |
| **Introduce GDE Service Deployment Component** |

The deployment option of the GDE allows the service developer to deploy the implemented grid service, which has been created with Introduce, to a Grid service container. The toolkit currently supports deploying a service to either a Globus, Tomcat, or JBoss Grid service container; however, support for other deployment options can easily be added to the GDE. The **General Deployment** tab shows basic information about the service you have going to deploy and provides the ability to choose the container to deploy to. Introduce will detect the available containers on the host machine by checking for the existing of environment variables (GLOBUS\_LOCATION (Globus), CATALINA\_HOME (Tomcat), JBOSS\_HOME (JBoss)). The **Advanced Deployment** tab enables configuration of many standard deployment options:

|  |  |  |
| --- | --- | --- |
| **Property Name** | **Value(s)** | **Description** |
| perform.index.service.registration | true of false | Whether or not the service should register with the Index Service |
| index.service.url | URL | The URL of the Index Service to register with. |
| index.service.index.refresh\_milliseconds | Integer | How often to reregister with the index service (this should be a relatively large amount of time and is simply useful for making sure the index service does not loose your registration). |
| index.service.registration.refresh\_seconds | Integer | When registering to the index service this number tell the index service how often, in milliseconds, to contact me for updated met |

The *Service Properties* deployment tab allows the service deployer to populate service configuration properties, which the service will have access to at runtime.

**Undeployment**

|  |
| --- |
| [[https://wiki.cagrid.org/download/attachments/2917693/undeployServiceScreen.png](https://wiki.cagrid.org/download/attachments/2917693/undeployServiceScreen.png?version=1)[https://wiki.cagrid.org/images/icons/link_attachment_7.gif](https://wiki.cagrid.org/download/attachments/2917693/undeployServiceScreen.png?version=1)](https://wiki.cagrid.org/download/attachments/2917693/undeployServiceScreen.png?version=1) |
| **Introduce GDE Service Undeployment Component** |

Services generated with Introduce 1.2 and higher support undeployment. The service, when deployed, creates a log file, containing information about what was copied into the container. Utilizing this information the undeployment task of the service is able to determine what was copied to the container that is not shared by any other service and it will remove it. This feature enables keeping the container from getting corrupted with left over jars and schema from deployments.

**Software Updates**

|  |
| --- |
| [[https://wiki.cagrid.org/download/attachments/2917693/softwareUpdateScreen.png](https://wiki.cagrid.org/download/attachments/2917693/softwareUpdateScreen.png?version=1)[https://wiki.cagrid.org/images/icons/link_attachment_7.gif](https://wiki.cagrid.org/download/attachments/2917693/softwareUpdateScreen.png?version=1)](https://wiki.cagrid.org/download/attachments/2917693/softwareUpdateScreen.png?version=1) |
| **Introduce Software Update** |

Introduce is capable of downloading and installing new extensions, upgrades to older extensions, and newer versions of itself. In the GDE there is a *Help* menu. In this menu there is a *Check for Updates* button. This button will take the user to a wizard which will walk them through looking for any software updates or new packages which they may want to download and install. The user can also put in a different software update site URL to point to a custom site containing Introduce extensions.

**Service Development**

**Service Upgrading**

Introduce has the ability to help upgrade a service to a newer version of Introduce. If the developer attempts to open a service generated with an older version of Introduce (1.0 and newer), Introduce will prompt the user to proceed with the migration process. The migration process is fully automated and when it is complete will report out to the developer what might be left for them to adjust based on there potentially custom changes or if there were any errors during the process.



When using Introduce to open a service for modification it will check the service to see which version of Introduce and its Extensions were used to create/modify the service. If those versions are different from those installed in the Introduce being used, it will prompt the user and notify them that the service needs upgrading. When prompted the user will have to decide to either:

* **Upgrade**: upgrade the service to the version that Introduce can properly work with it
* **Open**: attempt to have Introduce work with it without upgrading which is potentially dangerous and recommended.
* **Close**: do nothing to the service and do not proceed with the modification process

If the user chooses to upgrade the service the upgrade process will begin. Once finished, a report indicating the major changes and potential issues will be displayed to the user for their review. Once they are confident they have addressed any issues in the upgrade report, they can select the "Proceed" option and the Modification Viewer will be opened displaying the newly upgraded service. If the report warns them about potential modification they might need to make in order to finish the Upgrade process then the user can select the "Edit" button. This will enable them to halt the upgrade process while they make changes. In doing so, they can open the service up for modification at a later time and Introduce should be able to work with the service. If they are not confident in the changes and don't want to upgrade, they can select the "Roll Back" option to restore the service to its previous state.

**Debugging my service**

[[caGrid debugging article[https://wiki.cagrid.org/images/icons/linkext7.gif](http://www.cagrid.org/wiki/CaGrid:How-To:Debug)](http://www.cagrid.org/wiki/CaGrid:How-To:Debug)]

**Using the Client**

Introduce generates a client API for the service which is exactly as described within the graphical editing environment (see Auto-boxing/Unboxing or Service Operations). This client API can be used in order to leverage this type of service from another application or service. The API contains four constructors which can use used, each of which is different depending on having a handle or just an address and the need for security to be used.

/\*\*

\* Takes in the url of the service to connect to as a string

\*/

HelloWorldClient(String url)

/\*\*

\* Takes in the url of the service to connect to as a string and

\* a proxy to be used to represent the credentials or the caller

\*/

HelloWorldClient(String url, GlobusCredential proxy)

/\*\*

\* Takes in the epr which refers to the service or resource

\*/

HelloWorldClient(EndpointReferenceType epr)

/\*\*

\* Takes in the epr which refers to the service or resource and

\* a proxy to be used to represent the credentials or the caller

\*/

HelloWorldClient(EndpointReferenceType epr, GlobusCredential proxy)

Once a client handle is constructed each of the operations which were created in the service are available as operations to this newly constructed client instance. Below is an example snippet of code which creates a new client handle to a service called "HelloWorld" and calls the "echo" operation.

try {

HelloWorldClient client = new HelloWorldClient("http://localhost:8080/wsrf/services/HelloWorld");

client.echo("Testing)";

} catch (Exception e) {

System.out.println("Problem creating handle to or calling service" + e.getMessage(), e);

}

**Advance client usage**

**Secure Client Usage**

Introduce generated client code can utilize a certificate to communicate with its service securely. This certificate can come from many places; it may be a user certificate or a host certificate or some other certificate received from a delegation service for example. The client by default will attempt to use the credential in the default location in the users home directory. So for example, if you logged in using Dorian the credential would be written to the file system in the default location and your client would automatically use that credential when required by the service. If you want the client to use a different certificate you must pass that certificate into the constructor of the client or by calling the setProxy operation on the client. When the client makes a call to the service it will check the security metadata which tells the client how to configure itself so that it can properly communicate with the service. Even though a service's method is set to require secure communication, this does not mean the client will always use its own credentials. Introduce generated clients by default will connect anonymously to methods that allow both anonymous and non-anonymous access. If you want your client to use its credentials to invoke a method, even though that method can be invoked anonymously, you can set the client to prefer not connecting anonymously. This will force the client to use its own credentials to communicate with the service as opposed to connecting anonymously. In order to do this you must call the setAnonymousPrefered operation on the client you are using:

client.setAnonymousPrefered(false);

The client will then connect with credentials always until you set this back to true letting the client know it is ok to connect anonymously to methods that allow anonymous users. The reason for having this capability is because there may be methods that change the way they work based on who they are talking to. If they are talking to an anonymous user they may not return all the data and if the user has authenticated using their credentials than maybe they get back more privileged information.

You can also change the proxy (the credentials) that your client is using by calling the setProxy operation and passing in the new credentials you now want to use.

client.setProxy(newCredentials);

**Printing detailed faults**

The example below shows how a client can print a more detailed stack trace from an exception that is received from the service.

try {

...

} catch (Exception e) {

gov.nih.nci.cagrid.common.FaultUtil.printFault(e);

}

**Using Subscription and Notification**

If the Introduce service you are using supports the Notification resource framework option than the client will be enabled for subscribing to that service and listening for particular changes to resource properties. An example below shows how to perform the subscription:

IntroduceTestNotificationServiceClient client = new IntroduceTestNotificationServiceClient("http://localhost:8080/wsrf/services/IntroduceTestNotificationServiceClient");

client.subscribe(new QName("http://testing.org","TestResourceProperty");

In the above code we have an example client that is subscribing to a service to listen for changes to the TestResourceProperty resource property of the service. If the value of this resource property gets changed in the resource a notification will be sent out to all registered subscribers. Now we need to implement some code so that we receive these notifications. The introduce client generated for this service will automatically be able to listen for these changes. For us to be able to perform our own actions when we receive a notification we must overload the deliver operation in our client:

public void deliver(List topicPath, EndpointReferenceType producer, Object message) {

org.oasis.wsrf.properties.ResourcePropertyValueChangeNotificationType changeMessage = ((org.globus.wsrf.core.notification.ResourcePropertyValueChangeNotificationElementType) message)

.getResourcePropertyValueChangeNotification();

if (changeMessage != null) {

recievedNotificationCount++;

try {

System.out.println("GOT NOTIFICATION: " + changeMessage.getNewValue().get\_any()[0].getAsString());

} catch (Exception e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

}

Another way to subscribe is to implement a NotificationCallback class and pass an instance into the subscribe method as show below:

IntroduceTestNotificationServiceClient client = new IntroduceTestNotificationServiceClient("http://localhost:8080/wsrf/services/IntroduceTestNotificationServiceClient");

NotificationCallback callback = new MyNotificationCallback();

client.subscribe(new QName("http://testing.org","TestResourceProperty",callback);

This will enable the client to send notification back to the provided callback as opposed to all messages coming back to this client class and expecting the local implementation of the *deliver* method. This is useful for a more centralized message handling approach for when you might be getting notifications back from many different clients.

**Loading a proxy from a file**

If you want to use a specific proxy file, other than the locations that Globus will automatically look. The ProxyUtil class can be used to load the proxy and create a GlobusCredential.

GlobusCredential creds = null;

try {

creds = gov.nih.nci.cagrid.common.security.ProxyUtil.loadProxy("user.proxy");

System.out.println("Using proxy with id= " + creds.getIdentity() + " and lifetime "

+ creds.getTimeLeft());

} catch (Exception e1) {

System.out.println("No proxy file loaded so running with no credentials");

}

**Implementing the Service**

When an operation is added in the Introduce GDE and hte save button is clicked. Introduce will add the new stubbed method into the <service package>.service.<service name>Impl.java class. The developer is then responsible for implementing the method prior to deployment. For example the snippet below would be generated in the <service package>.service.<service name>Impl.java if the developer added an *add* operation to the service with Introduce that took in two integers and returned and integer.

public int add(java.math.BigInteger a,java.math.BigInteger b) throws RemoteException {

//TODO: Implement this autogenerated method

throw new RemoteException("Not yet implemented");

}

The developer would then have to edit this method to implement the logic the service would execute on invokation.

public int add(java.math.BigInteger a,java.math.BigInteger b) throws RemoteException {

return a + b;

}

**Advanced Service Development Topics**

**Obtaining the callers identity**

In the service it is sometime required to get the identity of the caller. The identity is carried in the context of the message which was passed into the service. In order to get this information the following lines will be needed.

String userDN = gov.nih.nci.cagrid.introduce.servicetools.security.SecurityUtils.getCallerIdentity()

**Accessing service properties**

Service properties are the properties that the main service can be configured with that all service contexts deployed with that service can access. These properties are configured at deploy time and will likely need to be accessed in the service. Below is example code for how to access these properties where the main service was named ***HelloWorld*** and the name of the service property was ***testing***.

HelloWorldConfiguration.getConfiguration().getTesting();

**Getting a Handle to a Resource**

When developing in the service context you must remember that the ServiceImpl class is just the service interface but the service should act on the addressed resource as there may be more than one resource instance being used by the service at a time. An example of how to get a handle to the resource which is being addressed by the current caller (exmaple from the counter service context as described in in the section on Utilizing the factory patter):

public void incrementCounter() throws RemoteException {

int currentCounter;

try {

currentCounter = getResourceHome().getAddressedResource().getIntValue();

currentCounter++;

getResourceHome().getAddressedResource().setIntValue(currentCounter);

} catch (Exception e) {

e.printStackTrace();

}

}

**Programmatically accessing and/or editing resource properties (metadata)**

From the resource there will be getters and setters automatically generated for the resource properties of your service. For instance, if you look at the above example, you can see that the counter service has an "int" resource property. This enables introduce to automatically generated getters and setters for that resource property that the service implementer can use to access and modify values in the resource properties.

**Creating a Security Descriptor for a Resource**

Sometimes it is necessary to set a security descriptor on an instance of a resource. This might be to protect resource instances so that only the creator of the resource can use it. In order to do this you have to make sure that when the context was generated in introduce that you had selected the "secure" resource framework option. This will enable the resource to have a setSecurityDescriptor() method enabled on it. When the resource is being created, as in the example below, the security descriptor should be generated and set on the resource. Once set it will be utilized on any access to the resource. Below is an example method from the counter service used in the utilizing a factory patter section. This is the method that Introduce generated to create the counter instance for the caller. You can see in this code that there is a place documenting how to set the security descriptor on the resource.

public example.counter.context.stubs.types.CounterContextReference createCounter() throws RemoteException {

org.apache.axis.message.addressing.EndpointReferenceType epr = new org.apache.axis.message.addressing.EndpointReferenceType();

example.counter.context.service.globus.resource.BaseResourceHome home = null;

org.globus.wsrf.ResourceKey resourceKey = null;

org.apache.axis.MessageContext ctx = org.apache.axis.MessageContext.getCurrentContext();

String servicePath = ctx.getTargetService();

String homeName = org.globus.wsrf.Constants.JNDI\_SERVICES\_BASE\_NAME + servicePath + "/" + "counterContextHome";

try {

javax.naming.Context initialContext = new javax.naming.InitialContext();

home = (example.counter.context.service.globus.resource.BaseResourceHome) initialContext.lookup(homeName);

resourceKey = home.createResource();

// Grab the newly created resource

example.counter.context.service.globus.resource.CounterContextResource thisResource = (example.counter.context.service.globus.resource.CounterContextResource)home.find(resourceKey);

// This is where the creator of this resource type can set whatever needs

// to be set on the resource so that it can function appropriatly for instance

// if you want the resouce to only have the query string then there is where you would

// give it the query string.

// sample of setting creator only security. This will only allow the caller that created

// this resource to be able to use it.

//thisResource.setSecurityDescriptor(gov.nih.nci.cagrid.introduce.servicetools.security.SecurityUtils.createCreatorOnlyResourceSecurityDescriptor());

String transportURL = (String) ctx.getProperty(org.apache.axis.MessageContext.TRANS\_URL);

transportURL = transportURL.substring(0,transportURL.lastIndexOf('/') +1 );

transportURL += "CounterContext";

epr = org.globus.wsrf.utils.AddressingUtils.createEndpointReference(transportURL,resourceKey);

} catch (Exception e) \

throw new RemoteException("Error looking up CounterContext home:" + e.getMessage(), e);

}

//return the typed EPR

example.counter.context.stubs.types.CounterContextReference ref = new example.counter.context.stubs.types.CounterContextReference();

ref.setEndpointReference(epr);

return ref;

}

**Setting the Termination Time on a Resource**

When the lifetime resource framework option is set on a service the resources of that service will be able to have tier lifetime managed. From the client side a SetTerminationTime() or Destroy() grid call can be made to the service either setting the time to terminate or destroying the addressed resource. You can also call the set termination time directly on the resource inside the service. An example below, utilizing the same code as above will enable us create counters for users that will only be alive for 5 minutes:

public example.counter.context.stubs.types.CounterContextReference createCounter() throws RemoteException {

org.apache.axis.message.addressing.EndpointReferenceType epr = new org.apache.axis.message.addressing.EndpointReferenceType();

example.counter.context.service.globus.resource.BaseResourceHome home = null;

org.globus.wsrf.ResourceKey resourceKey = null;

org.apache.axis.MessageContext ctx = org.apache.axis.MessageContext.getCurrentContext();

String servicePath = ctx.getTargetService();

String homeName = org.globus.wsrf.Constants.JNDI\_SERVICES\_BASE\_NAME + servicePath + "/" + "counterContextHome";

try {

javax.naming.Context initialContext = new javax.naming.InitialContext();

home = (example.counter.context.service.globus.resource.BaseResourceHome) initialContext.lookup(homeName);

resourceKey = home.createResource();

// Grab the newly created resource

example.counter.context.service.globus.resource.CounterContextResource thisResource = (example.counter.context.service.globus.resource.CounterContextResource)home.find(resourceKey);

// This is where the creator of this resource type can set whatever needs

// to be set on the resource so that it can function appropriatly for instance

// if you want the resouce to only have the query string then there is where you would

// give it the query string.

// sample of setting creator only security. This will only allow the caller that created

// this resource to be able to use it.

//thisResource.setSecurityDescriptor(gov.nih.nci.cagrid.introduce.servicetools.security.SecurityUtils.createCreatorOnlyResourceSecurityDescriptor());

//set the termination time of this resource

Calendar cal = new GregorianCalendar();

cal.add(Calendar.MINUTE,5);

thisResource.setTerminationTime(cal);

String transportURL = (String) ctx.getProperty(org.apache.axis.MessageContext.TRANS\_URL);

transportURL = transportURL.substring(0,transportURL.lastIndexOf('/') +1 );

transportURL += "CounterContext";

epr = org.globus.wsrf.utils.AddressingUtils.createEndpointReference(transportURL,resourceKey);

} catch (Exception e) {

throw new RemoteException("Error looking up CounterContext home:" + e.getMessage(), e);

}

//return the typed EPR

example.counter.context.stubs.types.CounterContextReference ref = new example.counter.context.stubs.types.CounterContextReference();

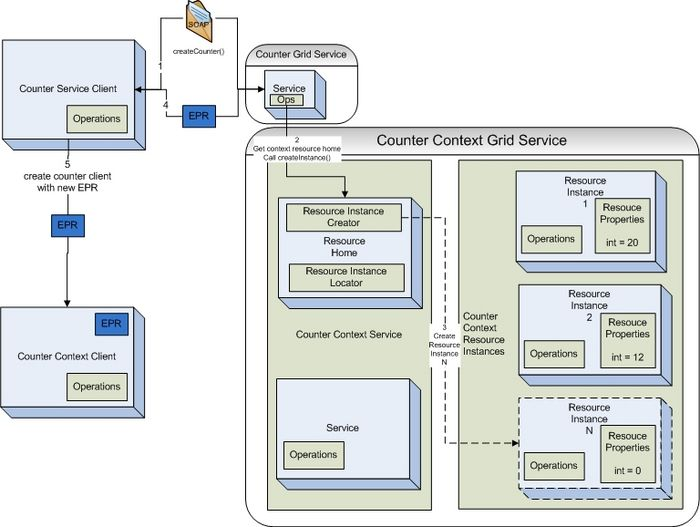
ref.setEndpointReference(epr);

return ref;

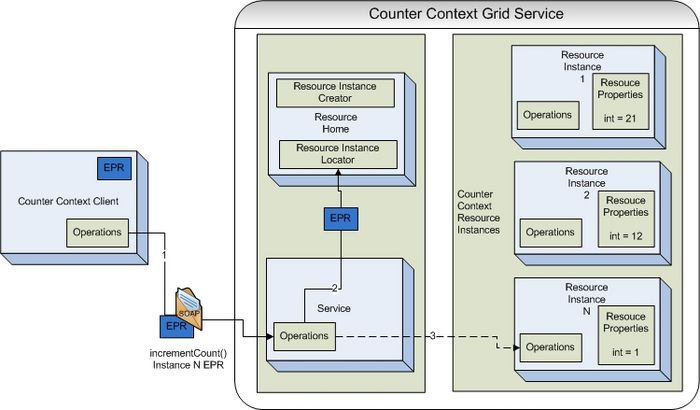
}

**Utilizing the Factory Pattern**

[Link to the entire example service source code described below[https://wiki.cagrid.org/images/icons/linkext7.gif](https://project.bmi.ohio-state.edu/gf/download/frsrelease/106/278/CounterServiceExample.zip)](https://project.bmi.ohio-state.edu/gf/download/frsrelease/106/278/CounterServiceExample.zip)



Utilizing Grid Service Factory Pattern  
In order to create instances of a resource something must tell the resource home of the service to create them. This is typically done using the factory pattern. A service or method is responsible for telling the resource to create a new instance of a resource type for the service. An example of the factory pattern in action is show it the image above. A factory service, the Counter Grid Service, contains a create operation which is exposed as a grid service method. When the client invokes this method the Counter Grid Service will locate the resource home of the Counter Context Grid Service and ask it to create a new resource instance. Once this resource intance is created the resource home will return a pointer or address to this resource instance called an EndPointReference (EPR). This EPR is then returned from the create method back to the client so that it has a pointer to these new resource that it can act on.  
In order to act on this resource it will need to construct a client which can talk to the grid service which represents this resource type. The Counter Context Client will be constructed using the EPR to address the specific resource context for the service. Next the user would like to call the *incrementCounter()* operation on the Counter Context Grid Service. The client will make the call to the grid service and the service will then look at the EPR sent over by the client. The service will then take the EPR and give it to the resource home and ask for the resource instance back which is represented by the particular EPR. The the service will run it's logic to increment the counter, which in this case is represented as a resource property of the resource.



Invoking a Statefull Grid Services  
Below shows an example of the code that would be required to implement the *incrementCounter()* method in the CounterContextServiceImpl.java.

package example.counter.context.service;

import java.rmi.RemoteException;

/\*\*

\* TODO:I am the service side implementation class. IMPLEMENT AND DOCUMENT ME

\*

\* @created by Introduce Toolkit version 1.1

\*/

public class CounterContextImpl extends CounterContextImplBase {

public CounterContextImpl() throws RemoteException {

super();

}

public void incrementCounter() throws RemoteException {

int currentCounter;

try {

currentCounter = getResourceHome().getAddressedResource().getIntValue();

currentCounter++;

getResourceHome().getAddressedResource().setIntValue(currentCounter);

} catch (Exception e) \{

e.printStackTrace();

}

}

}

In Introduce 1.3, the "incrementCounter()" operation shown above is implemented in the service context implementation class (CounterContextImpl). The context operations implemented in this class are called each time a context client calls an operation. If the service context you are writing (as in the counter case above) maintains state for each context instance between calls, then that state can be accessed via the "getResourceHome().getAddressedResource()" call shown above. This call returns the specific instance of a CounterContextResource class that holds the state for the context client. In summary, the CounterContextImpl class is a stateless class that simply implements the context operations. The CounterContextImpl class uses the CounterContextResource instance specific to the EPR used by the client to access and maintain state for the client who is currently calling the context operation. In the above example, the state in the CounterContextResource is the current count retrieved in the impl via the "getIntValue" method (which is a service context property added via Introduce GDE).

**Deserialization and Serialization Examples**

Introduce comes with helper tools for serializing and deserializing objects.This can be usefull when needing to read or write the data objects recieved from the grid to or from the filesystem or some other input/output mechanism.

**Deserialization**

The example below if the the cagrid transfer service. This example is deserializing an xml file into a TranasferServiceContextResourceProperties object.

TransferServiceContextResourceProperties props = null;

try {

props = (TransferServiceContextResourceProperties) Utils.deserializeObject(new FileReader(persistenceDir

+ File.separator + requestedID + ".xml"), TransferServiceContextResourceProperties.class);

} catch (Exception e) {

logger.info("Cannot find or deserialize the resource properties describing this transfer object: "

+ requestedID);

e.printStackTrace();

}

**Serialization**

Write the object out that we serialized above. This will use the QName of the schema element that this object adheres to and write the object out to the *test.xml* file.

try {

Utils.serializeDocument("test.xml", props, new QName("http://transfer.cagrid.org/TransferService/Context", "TransferServiceContextResourceProperties"));

} catch (Exception e1) \{

// TODO Auto-generated catch block

e1.printStackTrace();

}