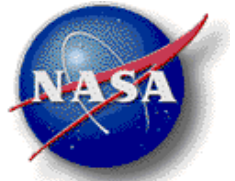


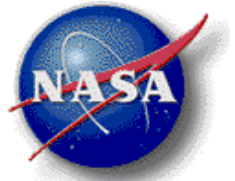
# Collaborative Infrastructure Tutorial Java

Ron van Hoof  
version 1.1  
22 April 2008

# Agenda

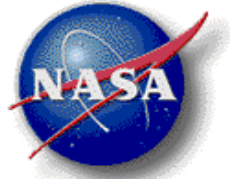


- Actor
- Actor Descriptor
- Actor Implementation
- Directory Service
- Transport Service
- Data Distribution Service
- Translation Service
- Management Service
- Time Service
- Logging Service
- Actor Hosting Environment
- Actor Service Provider
- Process Manager



- Interface to a component
- Controls component state
- Provides component status
- Access to services:
  - to publish itself and lookup other actors
  - to communicate with other actors
  - to publish data
  - to obtain time
  - to log data
- Two parts:
  - Actor Descriptor
  - Actor Implementation

# Actor Descriptor

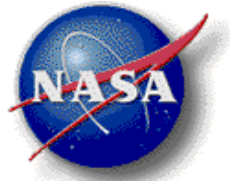


- Describes the actor, defines the actor's configuration settings and is used to load and configure the actor

```
<?xml version="1.0" encoding="UTF-8"?>  
<ACTOR>  
</ACTOR>
```

- Content consists of
  - Library
  - Credentials
  - Advertisement
  - Transport (optional)
  - Properties
- File extension: **.ciax**
- Contents available to Actor in ActorDescriptor struct

# Actor Descriptor - Library



- Defines where the implementation can be found

- Java Implementation (class name)

```
<LIBRARY type="ci.actor.class">  
  gov.nasa.ci.examples.dsa.DecisionSupportAgent  
</LIBRARY>
```

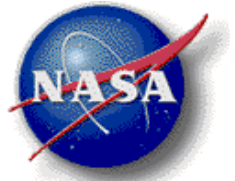
- Class and any support classes must be in the actor hosting environment's classpath.

- C++ Implementation (library file)

```
<LIBRARY type="ci.actor.library">  
  /usr/lib/libexecutive.so  
</LIBRARY>
```

- Simple file name can be specified as well, file must then be in the LD\_LIBRARY\_PATH (\*nix) or in the system path (Windows).

```
<LIBRARY type="ci.actor.library">  
  executive  
</LIBRARY>
```



- Specifies Actor's identifying information

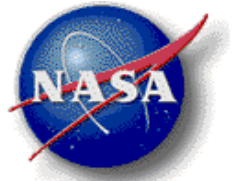
```
<CREDENTIALS>
  <SIMPLENAME>PowerExecutive</SIMPLENAME>

  <QUALIFIEDNAME>gov.nasa.executive.PowerExecutive</QUALIFIEDNAME>

  <DIRECTORYNAMES>
    <DIRECTORYNAME>gov.nasa.executive.PowerExecutive</DIRECTORYNAME>
    <DIRECTORYNAME>A40/executive/PowerExecutive</DIRECTORYNAME>
  </DIRECTORYNAMES>

  <LANGUAGES>
    <LANGUAGE>plexil</LANGUAGE>
  </LANGUAGES>

  <PROPERTIES>
    <PROPERTY name="some.property">some value</PROPERTY>
  </PROPERTIES>
</CREDENTIALS>
```



- Specifies Actor's capabilities

```
<ADVERTISEMENT>
```

```
<CAPABILITIES>
```

```
<CAPABILITY>
```

```
<NAME>Execution</NAME>
```

```
<KEYWORDS>
```

```
<KEYWORD>Execution</KEYWORD>
```

```
<KEYWORD>Power</KEYWORD>
```

```
<KEYWORD>Plan</KEYWORD>
```

```
</KEYWORDS>
```

```
<DESCRIPTION>
```

```
The Power Executive supports the execution of power related plans.
```

```
</DESCRIPTION>
```

```
</CAPABILITY>
```

```
</CAPABILITIES>
```

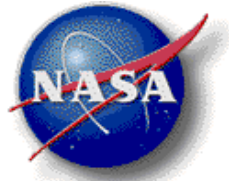
```
<PROPERTIES>
```

```
<PROPERTY name="some.property">some value</PROPERTY>
```

```
</PROPERTIES>
```

```
</ADVERTISEMENT>
```

# Actor Descriptor - Transport



## Automation for Operations

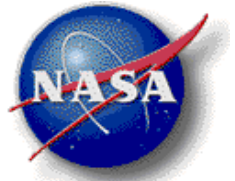
- Specifies how an actor can be reached (dedicated endpoints)
- Preferred approach is to configure only endpoints in the transport service descriptor for the actor service provider (using shared endpoints). Endpoints in the actor's descriptor are dedicated to the actor only.

```
<TRANSPORT>
  <ENDPOINTS>
    <ENDPOINT>
      <PROTOCOL>TCP</PROTOCOL>
      <PROPERTIES>
        <PROPERTY name="tcp.host">planware</PROPERTY>
        <PROPERTY name="tcp.port">0</PROPERTY>
      </PROPERTIES>
    </ENDPOINT>

    <ENDPOINT>
      <PROTOCOL>UDP</PROTOCOL>
      <PROPERTIES>
        <PROPERTY name="udp.host">planware</PROPERTY>
        <PROPERTY name="udp.port">0</PROPERTY>
      </PROPERTIES>
    </ENDPOINT>
  </ENDPOINTS>

  <PROPERTIES>
  </PROPERTIES>
</TRANSPORT>
```

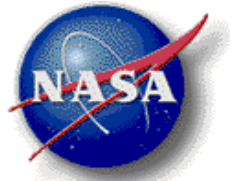
# Actor Descriptor - Properties



- Used to define other Actor configuration properties

```
<PROPERTIES>  
  <PROPERTY name="ci.actor.heartbeat.interval">2500</PROPERTY>  
  <PROPERTY name="my.config.file">PowerExecutive.cfg</PROPERTY>  
</PROPERTIES>
```

# Actor Descriptor IDL



- Actor Descriptor contents available to the Actor

```
struct ActorDescriptor {
    /** The Actor's Credentials as specified in the descriptor file */
    gov::nasa::ci::corba::api::Credentials ActorCredentials;

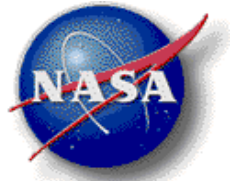
    /** The Actor's Advertisement as specified in the descriptor file */
    gov::nasa::ci::corba::api::directory::Advertisement ActorAdvertisement;

    /** The managed attributes declared for the actor in the descriptor file */
    gov::nasa::ci::corba::api::management::ManagedAttributeList ManagedAttributes;

    /** The managed operations declared for the actor in the descriptor file */
    gov::nasa::ci::corba::api::management::ManagedOperationList ManagedOperations;

    /** Other descriptor properties such as information about the
     * descriptor file itself */
    gov::nasa::ci::corba::api::util::PropertyList Properties;
}; // ActorDescriptor
```

# Actor Implementation



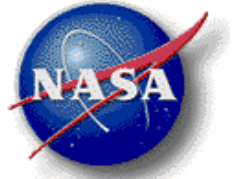
*Automation for Operations*

---

- IActor and AbstractActor
- Actor Information Methods
- Actor Service Methods
- Actor Control Methods
- Actor Status Methods
- Actor Message Processing
- Actor State Management
- Packaging
- Building
- Running the Actor

# Actor Implementation

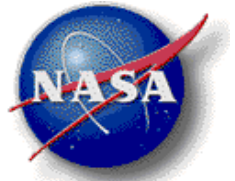
## Abstract Actor



- IActor – IDL interface for all actors
- AbstractActor – provides base implementation of all methods except, control, state/status and message processing, extends gov.nasa.ci.corba.api.IActorPOA
- Provides implementation to access Actor information, services, actor CORBA reference
- Java Implementation

```
import gov.nasa.ci.api.actor.AbstractActor;  
import org.apache.log4j.Logger;  
  
public class PowerExecutiveActor extends AbstractActor {  
    public final static Logger LOGGER = Logger.getLogger(PowerExecutiveActor.class);  
} // PowerExecutiveActor
```

# Actor Implementation Information Methods



- AbstractActor provides an implementation to access the actor's configuration information

```
public IActorDescriptor getDescriptor();
public ActorDescriptor getActorDescriptor();

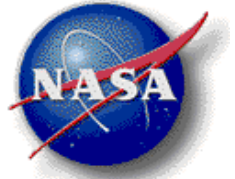
public IActor getReference() throws ActorException;

public String getSimpleName();
public String getQualified_name();
public String[] getDirectoryNames();

public Credentials getCredentials();
public void setCredentials(Credentials credentials);

public Advertisement getAdvertisement();
public void setAdvertisement(Advertisement advertisement);
```

# Actor Implementation Service Methods



*Automation for Operations*

---

- AbstractActor provides methods to access the services provided by the CI

```
public IDirectoryService getDirectoryService()  
    throws ServiceException;
```

```
public ITransportService getTransportService()  
    throws ServiceException;
```

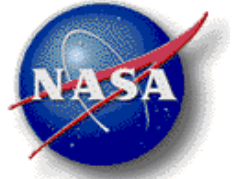
```
public IDataDistributionService getDataDistributionService()  
    throws ServiceException;
```

```
public IManagementService getManagementService()  
    throws ServiceException;
```

```
public ITimeService getTimeService()  
    throws ServiceException;
```

```
public ILoggingService getLoggingService()  
    throws ServiceException;
```

# Actor Implementation Control Methods



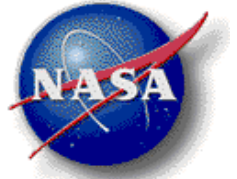
*Automation for Operations*

---

- Used to control the state of an actor
- Invoked by the Management Service
- Actor developer responsible for the implementation

```
public boolean isInitialized();  
  
public void initialize() throws ActorException;  
  
public void start() throws ActorException;  
  
public void suspend() throws ActorException;  
  
public void resume() throws ActorException;  
  
public void stop() throws ActorException;  
  
public void reset(CheckPoint cp) throws ActorException;  
  
public void shutdown();
```

# Actor Implementation Status Methods



*Automation for Operations*

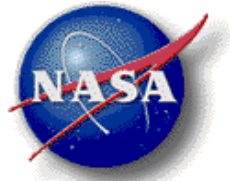
---

- Provides current state information about the actor
- Used by the Management Service to populate the heart beat
- Actor developer responsible for the implementation

```
public ActorStatus getStatus();
```

```
public ActorState getState();
```

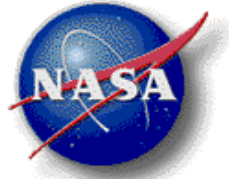
# Actor Implementation Message Processing



- Transport Service notifies the Actor of any messages to be handled by the Actor
- Actor developer responsible for the implementation

```
public void processMessage(ICommunicativeAct message);
```

# Actor Implementation State Management (1/2)



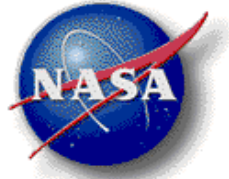
- The ActorState enumeration covers the following states:
  - created, initializing, initialized, starting, idle, waiting, executing, failing, suspending, suspended, resuming, stopping, stopped, resetting, finishing, finished, shutdown
- CI provides a State Manager
  - can optionally be used to manage the state, status and state transitions for an actor.
  - defines a state transition model
  - defines state transition and state related methods
- Method signatures

```
public void transitionTo(ActorState state) throws IllegalStateException;
public void transitionTo(ActorState state, long timeout) throws IllegalStateException;

public ActorState getState();
public ActorState getPreviousState();
public ActorStatus getStatus();
public void setCurrentTask(String task);

public boolean isInitialized();
public boolean inRunningState();
```

# Actor Implementation State Management (2/2)



## Automation for Operations

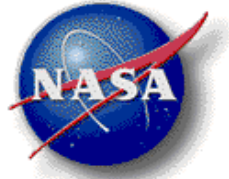
- Example use

```
void PowerExecutiveActor::start()
    throw (CORBA::SystemException, ActorException) {
    try {
        m_oStateManager.transitionTo(ActorState.STARTING);
        LOGGER.info(getSimpleName()+" - Starting..");
    } catch (IllegalStateException isx) {
        throw new ActorException(
            IExceptionHelper.getStackTrace(isx), IActorExceptionErrorCodes::INVALID_STATE_ERROR, "Invalid state change");
    } // end try

    // start
    try {
        // TODO: start the actor
    } catch (Throwable tx) {
        try {
            m_oStateManager.transitionTo(ActorState.FAILED);
            LOGGER.info(getSimpleName()+"(start) - Failed to start", tx);
        } catch (IllegalStateException isx) {
            // should never happen since any state can transition to FAILED
            LOGGER.fatal(getSimpleName()+" (start) - Unable to transition to FAILED state, should never happen!", isx);
        } // end try
        if (tx instanceof ActorException) {
            throw (ActorException)tx;
        } else {
            throw new ActorException(ExceptionHelper.getStackTrace(tx),
                IActorExceptionErrorCodes::UNEXPECTED_ERROR, "Unexpected error while starting.");
        } // end try
    } // end try

    // finalize start
    try {
        m_oStateManager.transitionTo(ActorState.IDLE);
    } catch (IllegalStateException isx) {
        throw new ActorException(
            ExceptionHelper.getStackTrace(isx), IActorExceptionErrorCodes::INVALID_STATE_ERROR, "Invalid state change");
    } // end try
} // start
```

# Actor Implementation Packaging



- Java: Actor's classes must be available on actor hosting environment's class path, preferably packaged as a jar file. Actor must have a default constructor, used by the CI to instantiate the actor.
- C++: Actor must be made available in a shared library/dynamically linked library
- Java – PowerExecutiveActor.java

```
package gov.nasa.executive;
```

```
import gov.nasa.ci.api.actor.AbstractActor;
```

```
import org.apache.log4j.Logger;
```

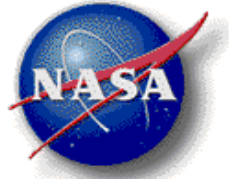
```
public class PowerExecutiveActor extends AbstractActor {
```

```
    public final static Logger LOGGER = Logger.getLogger(PowerExecutiveActor.class);
```

```
    ...
```

```
} // PowerExecutiveActor
```

# Actor Implementation Building (1/3)



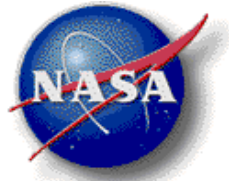
## Automation for Operations

- Use Ant (version 1.7.x) build script to compile and package the actor's classes.
- Template build script (build.xml) included with CI examples

```
<?xml version="1.0" encoding="UTF-8"?>
<project name="PowerExecutive" default="all" basedir=".">
  <property environment="env"/>
  <property file="build.properties"/>
  <property name="dir.src"          location="src"/>
  <property name="dir.build"       location="build"/>
  <property name="dir.dist"       location="dist"/>
  <property name="dir.ci_java_root" value="$(env.CI_JAVA_ROOT)"/>
  <property name="jar.ciinterface" value="${dir.ci_java_root}/dist/ciinterface.jar"/>
  <property name="jar.jacorb"     value="$(dir.ci_java_root)/lib/jacorb/jacorb.jar"/>
  <property name="jar.jacorb.logkit" value="$(dir.ci_java_root)/lib/jacorb/logkit-1.2.jar"/>
  <property name="jar.log4j"     value="$(dir.ci_java_root)/lib/log4j/log4j-1.2.14.jar"/>
  <property name="lib.jarfile"   value="powerexecutive.jar"/>

  <path id="classpath.common">
    <pathelement location="${jar.ciinterface}"/>
    <pathelement location="${jar.jacorb}"/>
    <pathelement location="${jar.log4j}"/>
  </path>
```

# Actor Implementation Building (2/3)



## Automation for Operations

```
<path id="classpath.common">
  <pathelement location="${jar.ciinterface}"/>
  <pathelement location="${jar.jacorb}"/>
  <pathelement location="${jar.log4j}"/>
</path>

<taskdef name="buildupdate" classname="gov.nasa.arc.ant.BuildUpdate"/>

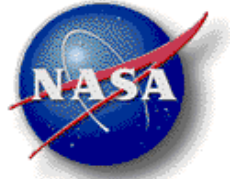
<target name="all" depends="init,init-release,compile,dist" description="Builds the Power Executive."/>
<target name="init">
  <tstamp/>
  <mkdir dir="${dir.build}"/>
</target>

<target name="init-release" depends="init" if="release.build" >
  <property name="build.debug" value="off"/>
</target>

<target name="build-number-update" depends="init-release" if="version.update" >
  <buildupdate srcdir="${dir.src}" classname="gov.nasa.executive.Version" variable="BUILD"/>
</target>

<target name="compile" depends="build-number-update"
  description="Compiles the source code located in ${dir.src} to ${dir.build}">
  <!-- By default include debug information -->
  <property name="build.debug" value="on"/>
  <!-- Compile the java code from ${dir.src} into ${dir.build} -->
  <javac srcdir="${dir.src}" destdir="${dir.build}" source="6" target="6" debug="${build.debug}">
    <classpath refid="classpath.common"/>
  </javac>
</target>
```

# Actor Implementation Building (3/3)



## Automation for Operations

```
<target name="dist" depends="compile"
  description="Creates the binary distribution in ${dir.dist}/${lib.jarfile}">
  <!-- Create the distribution directory -->
  <mkdir dir="${dir.dist}"/>
  <!-- Put everything in ${dir.build} into the ${lib.jarfile} file -->
  <jar jarfile="${dir.dist}/${lib.jarfile}" basedir="${dir.build}"/>
</target>

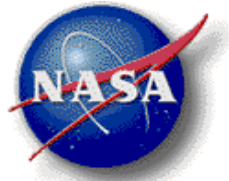
<target name="clean"
  description="Cleans up the output directories of the CI Examples - Executive">
  <delete dir="${dir.build}"/>
  <delete dir="${dir.dist}"/>
</target>

<target name="rebuild" depends="clean, compile"
  description="Cleans and rebuilds all sources for the CI Examples - Executive."/>
</project>
```

- To build the actor run ant in the directory of the build.xml file.
  - cd ~/Java/Executive
  - ant
- The jar file will be placed in the dist directory of the project.

# Actor Implementation

## Running the Actor



### Automation for Operations

---

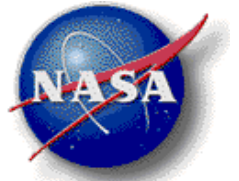
- Deploy the actor by copying its descriptor file (.ciax) into the actor hosting environment's deployment directory.
- Two possibilities to run the actor:
  1. Add the actor to the actor hosting environment's descriptor file in the <ACTORS></ACTORS> tag as:

```
<ACTOR>  
  <DESCRIPTOR>PowerExecutive.ciax</DESCRIPTOR>  
</ACTOR>
```

When the Actor Hosting Environment is started it will automatically load, initialize and start the actor.

2. Use the CI Console to request the actor hosting environment for the list of available actor descriptors. The new actor will be in that list. Request the actor hosting environment to load and initialize the actor. The actor will be listed in the CI Console at which point any of the management operations can be invoked on the actor including the start operation.

# Directory Service

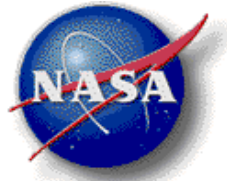


*Automation for Operations*

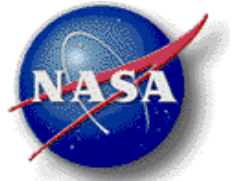
---

- Overview
- Configuration
- Naming
- Bind
- Rebind
- Lookup
- List Bindings
- Find (FY08)
- Unbind

# Directory Service Overview



- IDirectoryService interface with interfaces to:
  - register/deregister an actor using the actor's advertisement, supporting naming hierarchies
  - actor lookup by name or by capability, returning actor advertisements, components will never deal with direct references to actors, the credentials part of the advertisement are used in the envelope of the messages
- Directory service is distributed, each actor hosting environment has its own DS which discovers other DS's on the network and replicates directory entries between the DS's. DS's are discovered using IP multicasting
- Actor lookup by capability (FY08) allows for the support of high-availability features when multiple actors provide the same capabilities, one goes down, another with the same capability can be used.

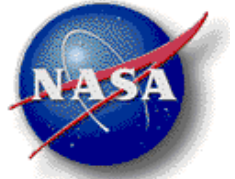


- Directory Service is an actor configured using a specialized actor descriptor (extension .cidx)

```
<DIRECTORY_SERVICE>  
</DIRECTORY_SERVICE>
```

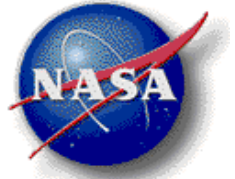
- Specifies
  - Credentials
  - Advertisement
  - Transport
  - Properties
- Two directory specific properties

# Directory Service Naming



- Uses a similar naming convention as CORBA name service
- Support for compound names (nested contexts)
- Name must not start with '/'
- Contexts are separated using '/'
- No support for the 'kind' attribute
- Examples:
  - gov.nasa.a4o.executive.PowerExecutive
  - nasa/a4o/executive/PowerExecutive
  - nasa/executive/gov.nasa.a4o.executive.PowerExecutive

# Directory Service Bind



- Bind is used to bind an advertisement to an unbound name. If an advertisement is already bound to that name an exception is raised
- CI automatically binds actors using the directory names in the actor's descriptor

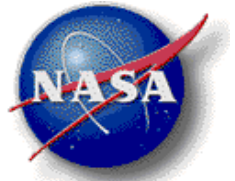
- Method signature:

```
void bind(String name, Advertisement actor)  
    throws InvalidNameException, NameAlreadyBoundException, NamingException,  
           ServiceException
```

- Example use:

```
try {  
    getDirectoryService().bind("a4o/executive/PowerExecutive", getAdvertisement());  
} catch (NameAlreadyBoundException nابخ) {  
    // a4o/executive/PowerExecutive already bound, use rebind to force binding  
} catch (Throwable tx) {}
```

# Directory Service Rebind



- Rebind is used to force the binding of an advertisement to a name. If an advertisement is already bound to that name then that binding is removed and updated with the new binding unless the binding is in a remote directory service.

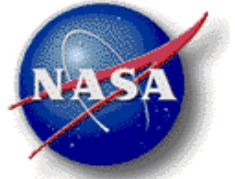
- Method signature:

```
void rebind(String name, Advertisement actor)
    throws InvalidNameException, NameAlreadyBoundException, NamingException,
           ServiceException
```

- Example use:

```
try {
    getDirectoryService().rebind("a4o/executive/PowerExecutive", getAdvertisement());
} catch (NameAlreadyBoundException nabx) {
    // a4o/executive/PowerExecutive bound in a remote directory
} catch (Throwable tx) {}
```

# Directory Service Lookup



- Lookup is used to retrieve an Actor's advertisement using the directory name of an actor. Lookup performs lookup in both the local directory service and proxies of the discovered remote directory services.

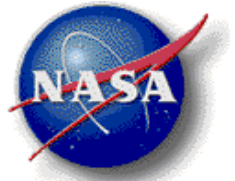
- Method signature:

```
Advertisement lookup(String name)
    throws InvalidNameException, NameNotFoundException, NamingException,
           ServiceException
```

- Example use:

```
try {
    Advertisement oAdvertisement =
        getDirectoryService().lookup("a4o/executive/PowerExecutive");
} catch (NameNotFoundException nnfx) {
    // a4o/executive/PowerExecutive not bound
} catch (Throwable tx) {}
```

# Directory Service List Bindings



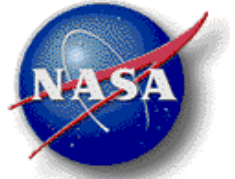
- List Bindings is used to retrieve a list of Actor advertisement bound in a naming context. Lookup performs lookup in both the local directory service and proxies of the discovered remote directory services.
- Method signature:

```
Advertisement[] listBindings(String name)
    throws InvalidNameException, NameNotFoundException, NamingException,
           ServiceException
```

- Example use:

```
try {
    Advertisement[] aoAdvertisements =
        getDirectoryService().listBindings("a4o/executive");
    LOGGER.debug(getSimpleName()+"(listBindings) - found "+aoAdvertisement.length+
        " bindings.");
} catch (NameNotFoundException nnfx) {
    // no bindings listed in the naming context a4o/executive
} catch (Throwable tx) {}
```

# Directory Service Unbind



- Unbind is used to unbind an advertisement from a bound name. If no advertisement is bound to that name the method silently succeeds
- CI automatically unbinds actors bound using the directory names in the actor's descriptor

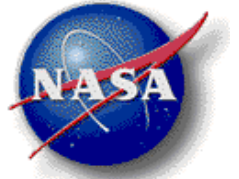
- Method signature:

```
void unbind(String name)  
    throws InvalidNameException, NamingException, ServiceException
```

- Example use:

```
try {  
    getDirectoryService().unbind("a4o/executive/PowerExecutive");  
} catch (Throwable tx) {}
```

# Transport Service

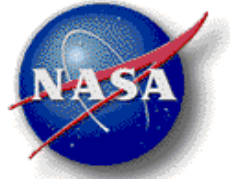


*Automation for Operations*

---

- Overview
- Configuration
- CommunicativeAct
- CommunicativeAct Factory
- Sending Message Asynchronously
- Sending Message Asynchronously with Response Handler
- Sending Message Synchronously
- Message Processing
- Sending Stream (FY08)
- Quality of Service Properties

# Transport Service Overview

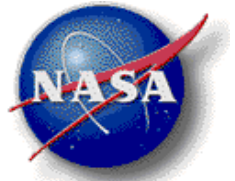


*Automation for Operations*

---

- Transport Service provides a message based communication service to communicate with other actors
- Messages are referred to as Communicative Acts and are based on the FIPA specification (Foundation for Intelligent Physical Agents – <http://www.fipa.org>)
- Support both asynchronous and synchronous message delivery
- Hides the transport implementation supporting a pluggable architecture. Actors can publish a variety of endpoints with different protocols declared in the actor's descriptor (TCP, UDP, Multicast, SSL, HTTP, IIOP, etc.)
- Currently supports TCP, SSL, UDP and Multicast endpoints

# Transport Service Configuration (1/2)



*Automation for Operations*

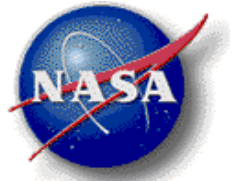
---

- A general transport service descriptor can be used to configure endpoints shared amongst all actors. No longer a need to configure a transport in each actor individually.
- Used for message based transport (Communicative Acts).
- Transport Service is configured using a specialized descriptor (extension .cicx)

```
<TRANSPORT_SERVICE>  
</TRANSPORT_SERVICE>
```

- **Specifies**
  - Endpoints
  - Properties
- Not used for data distribution service, it has a specialized transport layer and is still configured as part of the data distribution service's descriptor.

# Transport Service Configuration (2/2)



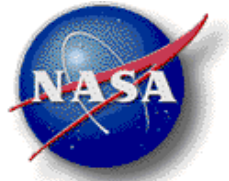
## Automation for Operations

```
<TRANSPORT_SERVICE>
  <ENDPOINTS>
    <ENDPOINT>
      <PROTOCOL>TCP</PROTOCOL>
      <PROPERTIES>
        <PROPERTY name="tcp.host">planware</PROPERTY>
        <PROPERTY name="tcp.port">0</PROPERTY>
      </PROPERTIES>
    </ENDPOINT>

    <ENDPOINT>
      <PROTOCOL>UDP</PROTOCOL>
      <PROPERTIES>
        <PROPERTY name="udp.host">planware</PROPERTY>
        <PROPERTY name="udp.port">0</PROPERTY>
      </PROPERTIES>
    </ENDPOINT>
  </ENDPOINTS>

  <PROPERTIES>
    <!-- many transport related configuration properties
    see the service template for a full listing. -->
  </PROPERTIES>
</TRANSPORT_SERVICE>
```

# Transport Service Communicative Act (1/3)

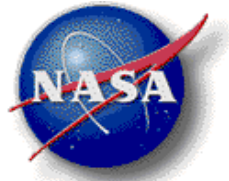


*Automation for Operations*

---

- Messaging based on FIPA Communicative Act's (<http://www.fipa.org/specs/fipa00037/>)
- Envelope specifies to/from (agent credentials, no direct agent references), QoS parameters, Security parameters, transport hints to allow the transport service to select the most appropriate transport.
- Payload specifies:
  - Performative (inform, request, subscribe, failure, abort, ...)
  - Conversation control (conversation id, reply-to, reply-with, in-reply-to, reply-by)
  - Language (DSA\_Plan, PLEXIL, PRL, ...), encoding, ontology
  - Message content (CORBA::Any) – any valid CORBA structures or CORBA primitive data types (long, long long, double, string, ...)

# Transport Service Communicative Act (2/3)



## Automation for Operations

- Communicative Act IDL

```
struct CommunicativeAct {
    CAEnvelope Envelope;
    CAPayload Payload;
}; // CommunicativeAct

struct CAEnvelope {
    long long CreationDate;
    PropertyList Envelope;
}; // CAEnvelope

struct CAPayload {
    PropertyList Payload;
}; // CAPayload

struct Property {
    string Name;
    any Value;
}; // Property

sequence<Property> PropertyList;
```

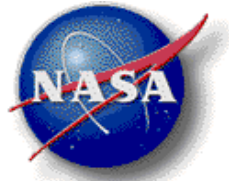
- Abstract interfaces define constants for property names

```
abstract interface ICAEnvelopeProperty {
    const string TO = "to";
};

abstract interface ICAPayloadProperty {
    const string PERFORMATIVE = "performative";
};
```

- Envelope requires **from** and **to** properties. Payload requires **performative** property. Other properties are optional.

# Transport Service Communicative Act (3/3)



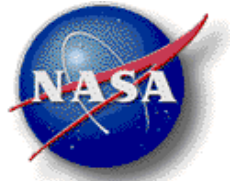
- CI provides interfaces and wrapper classes for the CommunicativeAct structure, envelope and payload with methods to access and set the properties using property specific methods to simplify the use of the CommunicativeAct and properly identifying the expected value types.

```
public interface ICommunicativeAct {           // implementation class CommunicativeActHelper
    public IEnvelope getEnvelope();
    public void setEnvelope(IEnvelope envelope);
    ...
    public CommunicativeAct getCommunicativeAct();
} // ICommunicativeAct
```

```
public interface IEnvelope {                 // implementation class EnvelopeHelper
    public Credentials[] getTo();
    public void setTo(Credentials[] recipients);
    public void addTo(Credentials recipient);
    public void replaceTo(Credentials oldRecipient, Credentials newRecipient);
    public void removeTo(Credentials recipient);
    ...
} // IEnvelope
```

```
public interface IPayload {}                // implementation class PayloadHelper
```

# Transport Service Communicative Act Factory



- Factory to simplify creation of Communicative Act's

```
public class CommunicativeActFactory {
    public static String createConversationID(Credentials creator);

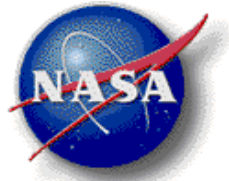
    public static ICommunicativeAct create(
        Credentials sender, Credentials receiver, Credentials replyTo,
        CAPerformative p, String language, String conversationID,
        String action, Any content);
    ...

    public static ICommunicativeAct createRequest(        // performative = REQUEST
        Credentials sender, Credentials receiver, Credentials replyTo,
        String language, String action, Any content);
    ...

    public static ICommunicativeAct createResponse(
        Credentials sender, Credentials receiver, CAPerformative p,
        String language, String action, Any content,
        ICommunicativeAct inResponseTo);
    ...
} // CommunicativeActFactory
```

# Transport Service

## Sending Message Asynchronously



- Used to send a message for which either no response is expected, to send a response to a message, or for which the response is handled by the actor's generic message handler

- Method signature

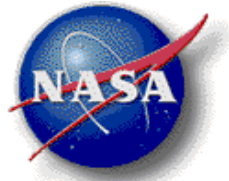
```
public void sendMessage(CommunicativeAct message)
    throws NotLocatableException, TransportFailure
```

- Example use

```
Credentials oSender = ...;
Credentials oReceiver = ...;
Any oContent = ORBHelper.getORB().create_any();
oContent.insert_string("Hello World!");
ICommunicativeAct oCA = CommunicativeActFactory.create(oSender, oReceiver,
    CAPerformative.INFORM, oContent);
try {
    getTransportService().sendMessage(oCA.getCommunicativeAct());
} catch (NotLocatableException nlx) {
    // receiver not reachable
} catch (Throwable tx) {
    // transmission or CORBA failure
} // end try
```

# Transport Service

## Sending Message w/ Handler (1/3)



### Automation for Operations

- Used to send a message for which a response is expected and the developer expects one or more responses to the message and wishes to handle them asynchronously.
- Message requires a conversation id (createRequest method generates one automatically)
- Method signatures

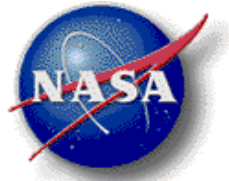
```
public int sendAsynchronousMessage(  
    CommunicativeAct message,  
    IMessageHandler handler)  
    throws NotLocatableException, TransportFailure;
```

- Example use

```
Credentials oSender = ...; // DSA  
Credentials oReceiver = ...; // Executive  
PlanStructure oPS = ...;  
Any oContent = ORBHelper.getORB().create_any();  
PlanStructureHelper.insert(oContent, oPS);  
// create the message  
ICommunicativeAct oCA = CommunicativeActFactory.createRequest(  
    oSender, oReceiver,  
    "PlanStructure", // content language  
    "addPlan", // action  
    oContent);  
  
// create the message handler  
MyHandlerImpl oHandler = new MyHandlerImpl();  
IMessageHandler oHandlerRef = oHandler._this(ORBHelper.getORB());  
  
// send the message  
int nID;  
try {  
    nID = getTransportService().sendAsynchronousMessage(oCA.getCommunicativeAct(), oHandlerRef);  
} catch (NotLocatableException nlx) {  
    // receiver not reachable  
} catch (Throwable tx) {  
    // transmission or CORBA failure  
} // end try
```

# Transport Service

## Sending Message w/ Handler (2/3)



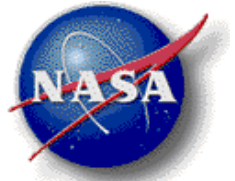
Automation for Operations

- Example use (cont'd)

```
// create the message handler
public class MyHandlerImpl extends IMessageHandlerPOA {
    public void processMessage(CommunicativeAct response, CommunicativeAct inResponseTo) {
        CommunicativeActHelper oCA = new CommunicativeActHelper(response);
        CAPerformative oPerformative = oCA.getPayload().getPerformative();
        switch (oPerformative) {
            case INFORM: {
                Any oContent = oCA.getPayload().getContent();
                try {
                    boolean bResult = oContent.extract_boolean();
                    if (bResult) {
                        // plan successfully added by the executive
                    } else {
                        // plan not added by the executive
                    } // end if bResult
                } catch (Throwable tx) {
                    // invalid response, expected boolean value
                } // end if extract boolean
                break;
            }
            case FAILURE: ...; break;
        } // end switch
    } // processMessage
}; // MyHandler_impl
```

# Transport Service

## Sending Message w/ Handler (3/3)



*Automation for Operations*

---

- Deregister the message handler when no more responses are expected

- Method signature

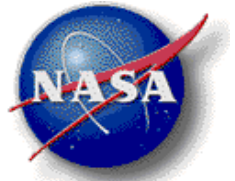
```
public void deregisterMessageHandler(int handlerID);
```

- Example use

```
try {  
    getTransportService().deregisterMessageHandler(nID);  
} catch (...) {  
} // end try
```

# Transport Service

## Sending Message Synchronously (1/2)



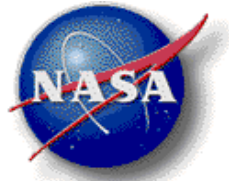
*Automation for Operations*

- Used to send a message for which only one response is expected and the developer wishes to block until that response is received or a time-out occurs.
- Message requires a conversation id
- The transport service generates a message handler for the caller
- Method signature

```
public int sendSynchronousMessage(  
    CommunicativeAct message,  
    CommunicativeActHolder response,  
    int timeout)  
    throws MessageNotRepliedToException, NotLocatableException, TransportFailure;
```

# Transport Service

## Sending Message Synchronously (2/2)



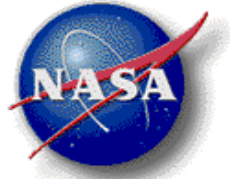
Automation for Operations

- Example use

```
Credentials oSender = ...; // DSA
Credentials oReceiver = ...; // Executive
PlanStructure oPS = ...;
Any oContent = ORBHelper.getORB().create_any();
PlanStructureHelper.insert(oContent, oPS);
// create the message
ICommunicativeAct oCA = CommunicativeActFactory.createRequest(
    oSender, oReceiver,
    "PlanStructure", // content language
    "addPlan", // action
    oContent);

// send the message
try {
    CommunicativeActHolder oResponseHolder = new CommunicativeActHolder();
    getTransportService().sendSynchronousMessage(oCA.getCommunicativeAct(), oResponseHolder, 10000);
    // process the response
    CommunicativeAct oResponse = oResponseHolder.value;
} catch (MessageNotRepliedToException mnrtx) {
    // timed out waiting for a response
} catch (NotLocatableException nlx) {
    // receiver not reachable
} catch (Throwable tx) {
    // transmission or CORBA failure
} // end try
```

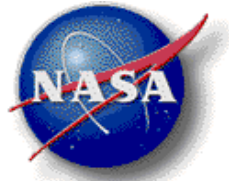
# Transport Service Message Processing (1/2)



- The Transport Service sends messages to the actor's message handler if the message is not:
  - a message for the management service
  - a response for which a handler is registered
  - a response to a synchronous message
- and if
  - the qualified name of one of the recipients for the message matches that of the actor
  - the message requires no translation or the message was translated into the language required by the actor
- Used generally to handle requests
- Method signature

```
public void processMessage(ICommunicativeAct message);
```

# Transport Service Message Processing (2/2)

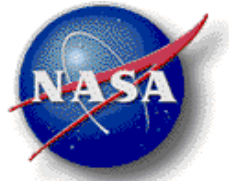


Automation for Operations

- Recommended approach is to use a message handler factory that retrieves handlers by performative and action
- Example implementation:

```
public void processMessage(ICommunicativeAct message) {
    CommunicativeActHandlerFactory oFactory = CommunicativeActHandlerFactory.Instance(this);
    ICommunicativeActHandler oHandler = oFactory.getHandler(message);
    if (oHandler != null) {
        oHandler->process(message);
    } else {
        // make sure we did not receive a NOT_UNDERSTOOD response for one of our messages
        CAPerformative oPerformative = message.getPayload().getPerformative();
        if (oPerformative != null && oPerformative == CAPerformative.NOT_UNDERSTOOD) {
            // not-understood response received
            return;
        } else {
            // no handler for the type of message, send not understood response back
            ICommunicativeAct oResponse = CommunicativeActFactory.createResponse(
                getCredentials(), CAPerformative.NOT_UNDERSTOOD, message);
            try {
                getTransportService().sendMessage(oResponse.getCommunicativeAct());
            } catch (Throwable tx) {
                ...
            } // end try
        } // end if
    } // end if
} // processMessage
```

# Transport Service Quality of Service Properties (1/2)

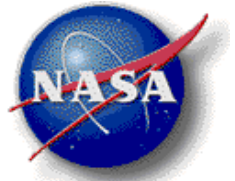


## Automation for Operations

- The QoS requirements for a message can be specified in its envelope
- CI currently supports only reliability property
  - `"ci.communication.qos.reliability"` or  
`gov.nasa.ci.api.communication.IQoSPropertyNames.QOS_RELIABILITY_PROPERTY`
- The reliability can have two possible values
  - “RELIABLE” – attempt to use a reliable transport to transmit the message (TCP/IP based), data delivery guaranteed provided the destination host/port can be reached.
  - “BEST EffORT” – attempt to use a best-effort transport to transmit the message (UDP based), data delivery is not guaranteed
- Default reliability for messages is: RELIABLE
- If the QoS is BEST EffORT and either the receiver publishes no such endpoint or the message delivery fails, the transport service will attempt to use a RELIABLE endpoint for delivery of the message

# Transport Service

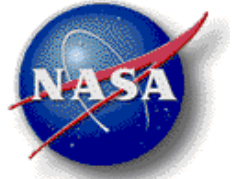
## Quality of Service Properties (2/2)



- Example use

```
CommunicativeAct oMessage = CommunicativeActFactory.createRequest(  
    getCredentials(),  
    oReceiver,  
    "PlanStructure",  
    "addPlan",  
    oContent);  
  
Any oQoSReliabilityValue = ORBHelper.getORB().create_any();  
oQoSReliabilityValue.insert_string(QoSReliability.BEST_EFFORT.toString());  
oMessage.getEnvelope().setPropertyValue(  
    IQoSPropertyNames.QOS_RELIABILITY_PROPERTY,  
    oQoSReliabilityValue);  
  
try {  
    getTransportService().sendMessage(oMessage.getCommunicativeAct());  
} catch (Throwable tx) {  
} // end try
```

# Data Distribution Service

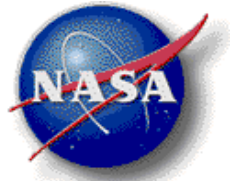


*Automation for Operations*

---

- Overview
- Data Info Publication
- Obtain Published Data Info
- Data Subscription
- Data Object
- Data Publication
- Quality of Service Properties
- Data Pull (FY08)

# Data Distribution Service Overview

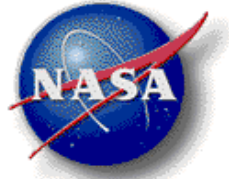


*Automation for Operations*

---

- Used to publish data where the sender is not interested in knowing who the subscribers are
- Used to subscribe for data where the origin of the data is not necessarily as important as the data itself
- Used to publish data from one publisher to many subscribers
- Used to subscribe for data of the same type published by many publishers
- Distributed, DDS' discover one another using IP multicasting and periodically publish for each type of data their publishers and subscribers
- DDS publishes only if subscribers are present

# Data Distribution Service Configuration (1/2)



## Automation for Operations

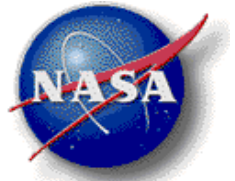
- Data Distribution Service is configured using a specialized actor descriptor (extension **.citx**)

```
<DATA_DISTRIBUTION_SERVICE>  
</DATA_DISTRIBUTION_SERVICE>
```

- Specifies
  - Credentials
  - Advertisement
  - Transport
  - Properties
- Transport requires multicast endpoint for publisher/subscriber discovery, address/port must be the same for all DDS' for every actor hosting environment that needs to communicate with one another

```
<ENDPOINT>  
  <PROTOCOL>MULTICAST</PROTOCOL>  
  <PROPERTIES>  
    <PROPERTY name="multicast.address">235.0.0.1</PROPERTY>  
    <PROPERTY name="multicast.port">5000</PROPERTY>  
    <PROPERTY name="multicast.buffersize">65535</PROPERTY>  
  </PROPERTIES>  
</ENDPOINT>
```

# Data Distribution Service Configuration (2/2)



- Two data distribution service specific properties

```
<PROPERTY  
name="ci.dds.advertisement.publication.interval">5000</PRO  
PERTY>
```

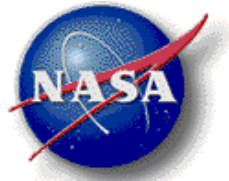
```
<PROPERTY  
name="ci.dds.subscription.expiration">15</PROPERTY>
```

- Reference to the descriptor is placed in the actor service provider's descriptor file

```
<DATA_DISTRIBUTION_SERVICE>  
  <DESCRIPTOR>HEPowerExecutive_DDS.citx</DESCRIPTOR>  
</DATA_DISTRIBUTION_SERVICE>
```

# Data Distribution Service

## Data Info Publication (1/2)



*Automation for Operations*

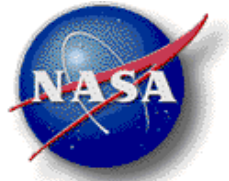
- Actor must indicate the types of data it intends to publish. Type of data is defined as a DataInfo struct (IDL):

```
struct DataInfo {  
    string Domain;  
    string Topic;  
    PropertyList QoSProperties;  
}; // DataInfo
```

- QoSProperties specifies the desired QoS properties for the publisher
- Method Signature

```
int addPublisher(DataInfo info)  
    throws ServiceException  
void changePublisher(int publisherID, DataInfo info)  
    throws DataTypeChangeException, ServiceException  
void removePublisher(int publisherID)  
    throws ServiceException
```

# Data Distribution Service Data Info Publication (2/2)



*Automation for Operations*

- Example use

```
try {
    IDataInfo oDataInfo = new DataInfoHelper(
        "Executive",    // Domain
        "ExecutionEvent" // Topic
    );

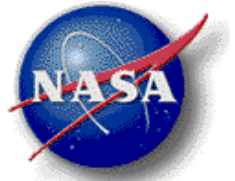
    // register for publication
    int nID = getDataDistributionService.addPublisher(oDataInfo.getDataInfo());

    // change QoS
    Any oQoSReliabilityValue = ORBHelper.getORB().create_any();
    oQoSReliabilityValue.insert_string(QoSReliability.RELIABLE.toString());
    Property oQoSReliability = new Property(
        IQoSPropertyNames::QOS_RELIABILITY_PROPERTY
        oQoSReliabilityValue);
    oDataInfo.addQoSProperty(oQoSReliability);
    getDataDistributionService().changePublisher(nID, oDataInfo.getDataInfo());

    // deregister the publication
    getDataDistributionService().removePublisher(nID);
} catch (Throwable tx) {}
```

# Data Distribution Service

## Obtain Published Data Info



*Automation for Operations*

---

- It is possible to request a list of all the types of data published by an actor

- **Method Signature**

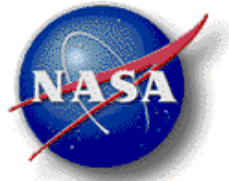
```
DataInfo[] getPublishedDataTypes(Credentials actor)  
    throws ServiceException;
```

- **Example use**

```
try {  
    DataInfo[] aoDataInfoList =  
        getDataDistributionService.getPublishedDataTypes(getCredentials());  
    for (DataInfo oDataInfo : aoDataInfoList) {  
        // do something with oDataInfo  
    } // end for  
} catch (Throwable tx) {}
```

# Data Distribution Service

## Data Subscription (1/3)



- To receive data an Actor must subscribe for that data specifying the type of data it wishes to subscribe for. Type of data is defined as a DataInfo struct (IDL):

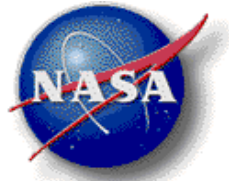
```
struct DataInfo {  
    string Domain;  
    string Topic;  
    PropertyList QoSProperties;  
}; // DataInfo
```

- QoSProperties specifies the desired QoS properties for the subscriber. A subscriber's QoS has precedence over that of the publisher's.
- Method Signature

```
int subscribe(DataInfo info, IDataSubscriber subscriber)  
    throws ServiceException  
int subscribeFrom(DataInfo info, Credentials actor, IDataSubscriber subscriber)  
    throws ServiceException  
void unsubscribe(int subscriberID)  
    throws ServiceException
```

# Data Distribution Service

## Data Subscription (2/3)



*Automation for Operations*

- Example use

```
try {
    IDataInfo oDataInfo = new DataInfoHelper(
        "Executive",          // Domain
        "ExecutionEvent"     // Topic
    );

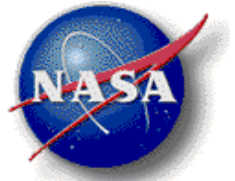
    // create subscriber
    MySubscriberImpl oSubscriber = new MySubscriberImpl();
    IDataSubscriber oSubscriberRef = oSubscriber._this(ORBHelper.getORB());

    // subscribe for data
    int nID1 = getDataDistributionService().subscribe(
        oDataInfo.getDataInfo(), oSubscriberRef);

    // subscribe for data from specific actor
    Credentials oFrom = ...;
    int nID2 = getDataDistributionService().subscribeFrom(
        oDataInfo.getDataInfo(), oSubscriberRef, oFrom);

    // unsubscribe for data
    getDataDistributionService().unsubscribe(nID2);
    getDataDistributionService().unsubscribe(nID1);

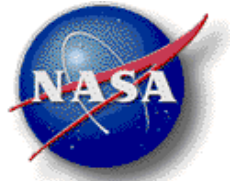
} catch (Throwable tx) {}
```



- Example use (cont'd)

```
class MySubscriberImpl extends IDataSubscriberPOA {  
    public void onDataReceipt(DataObject data) {  
        // process the DataObject  
        Any oContent = data.Content;  
        try {  
            String sContent = oContent.extract_string();  
            // process the XML Executive event  
        } catch (Throwable tx) {  
            // failed to extract the string contents  
        } // end try  
    } // onDataReceipt  
} // MySubscriber_impl
```

# Data Distribution Service Data Object



*Automation for Operations*

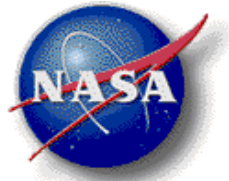
- Data to be published is defined in a DataObject struct (IDL)

```
struct DataObject {  
    string ID;  
    long long Timestamp;  
    string Domain;  
    string Topic;  
    Credentials Sender;  
    PropertyList FilterableData;  
    string Language;  
    any Content;  
}; // DataInfo
```

- The FilterableData can be populated but the DDS currently does not process it.

# Data Distribution Service

## Data Publication (1/2)



*Automation for Operations*

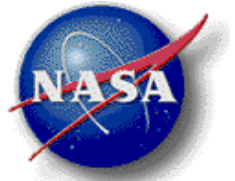
- An Actor can publish data at any time provided the actor indicated to the DDS that it published that data (a `PublicationException` is raised if this is not the case)
- Actors publish Data Objects using the DDS.
- The DDS determines if any subscribers have registered for the data, if so it publishes the data to those subscribers, if not the data won't be published.

- Method signature

```
void publishData(DataObject data)  
    throws PublicationException, ServiceException
```

- Example use

# Data Distribution Service Data Publication (2/2)



*Automation for Operations*

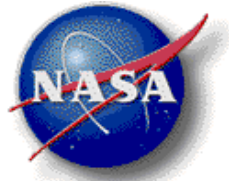
- Example use

```
try {
    Any oContent = ORBHelper.getORB().create_any();
    oContent.insert_string("<Event><EventId>1</EventId>...</Event>");
    // DataObjectHelper generates a unique ID and timestamp
    IDataObject oData = new DataObjectHelper(
        getSender(),           // sender
        "Executive",          // domain
        "ExecutionEvent",     // topic
        oContent,             // content
        "plexil"              // language
    );

    // publish the data
    getDataDistributionService().publishData(oData.getDataObject());
} catch (Throwable tx) {}
```

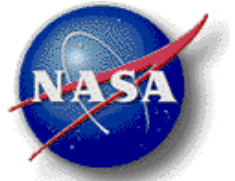
# Data Distribution Service

## Quality of Service Properties



- Both the publisher and subscriber can specify quality of service properties for the type of data they publish or subscribe to (DataInfo.QoSProperties)
- CI currently supports only reliability property  
`"ci.communication.qos.reliability"` or  
`gov.nasa.ci.api.communication.IQoSPropertyNames.QOS_RELIABILITY_PROPERTY`
- The reliability can have two possible values
  - “RELIABLE” – attempt to use a reliable transport to transmit the data (TCP/IP based), data delivery guaranteed provided the destination host/port can be reached.
  - “BEST\_EFFORT” – attempt to use a best-effort transport to transmit the data (UDP based), data delivery is not guaranteed
- Default reliability for data is: **BEST\_EFFORT**
- The DDS gives the subscriber’s QoS precedence over the publisher’s.

# Translation Service

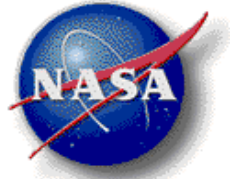


*Automation for Operations*

---

- Overview
- Message Content Translation
- Data Content Translation
- Translation Service Configuration
- Registering a Translator
- Writing a Translator

# Translation Service Overview



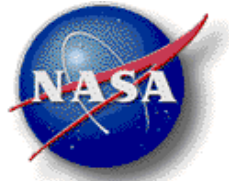
*Automation for Operations*

---

- Each actor hosting environment publishes one translation service.
- The translation service allows for automatic content translation of both message and data content
- Uses the language properties of a CommunicativeAct's payload and DataObject with the language specified in an actor's descriptor
- Actor developer does not interact directly with the translation service
- The translation service supports multiple translators
- Translators are described in translation service's descriptor
- Translators are written in Java

# Translation Service

## Message Content Translation



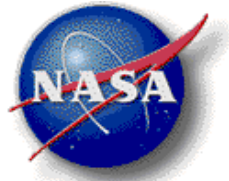
*Automation for Operations*

---

- A CommunicativeAct's payload can optionally specify a language (string).
- If a language is specified for the message and the intended recipient of the message specifies no language preference then no translation takes place and the message is delivered as is
- If a language is specified for the message and the intended recipient of the message specifies one or more languages then the message content is translated to the first language for the recipient for which translation succeeds, if none of the translations succeed, the message is not delivered to the recipient
- If no language is specified for the message and the recipient specifies one or more languages then no translation takes place and the message is delivered to the recipient as is. The recipient actor can attempt to interpret the content or return a not\_understood response to the sender.

# Translation Service

## Data Content Translation



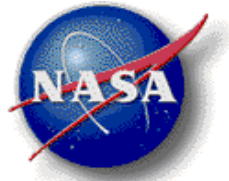
*Automation for Operations*

---

- A DataObject can optionally specify a language (string), empty string if no language is specified.
- If a language is specified for the data and the subscriber of the data specifies no language preference then no translation takes place and the data is delivered as is
- If a language is specified for the data and the subscriber of the data specifies one or more languages then the data content is translated to the first language for the subscriber for which translation succeeds, if none of the translations succeed, the data is not delivered to the subscriber
- If no language is specified for the data and the subscriber specifies one or more languages then no translation takes place and the data is delivered to the subscriber as is. The subscriber can attempt to interpret the content or discard the data

# Translation Service

## Translation Service Configuration



Automation for Operations

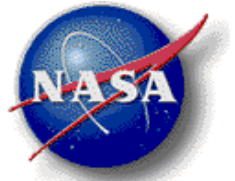
- The translation service descriptor describes the translators made available through the translation service
- The descriptor is placed in the actor service provider's configuration directory
- Extension of the translation service descriptor file is: **.cilx**

```
<?xml version="1.0" encoding="UTF-8"?>  
<TRANSLATION_SERVICE>  
  <TRANSLATORS>  
  </TRANSLATORS>  
</TRANSLATION_SERVICE>
```

- Add a **<TRANSLATION>** tag to the actor service provider's descriptor to identify the translation service descriptor file

```
<TRANSLATION_SERVICE>  
  <DESCRIPTOR>TranslationService.cilx</DESCRIPTOR>  
</TRANSLATION_SERVICE>
```

# Translation Service Registering a Translator

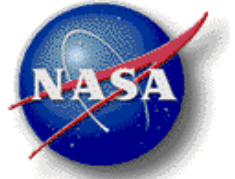


- For each translator add a <TRANSLATOR> tag to the translation service's descriptor.
- Language strings are case insensitive

```
<?xml version="1.0" encoding="UTF-8"?>
<TRANSLATION_SERVICE>
  <TRANSLATORS>
    <TRANSLATOR name="PlanStructurePlexilTranslator"
      class="gov.nasa.ci.examples.translators.
        PlanStructurePlexilTranslator">
      <TRANSLATES from="PlanStructure" to="PLEXIL"/>
      <TRANSLATES from="PLEXIL" to="PlanStructure"/>
    </TRANSLATOR>
  </TRANSLATORS>
</TRANSLATION_SERVICE>
```

# Translation Service

## Writing a Translator (1/3)



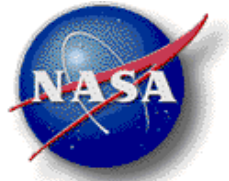
- Translators are developed in Java
- Translator must:
  - implement the ITranslator interface or
  - extend the AbstractTranslator class
- AbstractTranslator adds built-in support for configuring the translator extracting and setting the name and TranslationSupport from the translator's descriptor and providing an implementation for the `boolean translates(String from, String to)` method.

- Using AbstractTranslator implement:

```
public Any translateMessageContent(Any content, String from, String to)
    throws TranslationException;
public Any translateDataContent(IDataInfo info, Any content, String from, String to)
    throws TranslationException;
```

# Translation Service

## Writing a Translator (2/3)



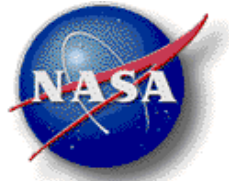
- Example

```
public class PlanStructurePlexilTranslator extends AbstractTranslator {
    public PlanStructurePlexilTranslator() {
        m_moPlanStructureToPlexilMapping = new HashMap<Integer, String>();
        m_moPlexilToPlanStructureMapping = new HashMap<String, Integer>();
    } // PlanStructurePlexilTranslator
    private Map<Integer, String> m_moPlanStructureToPlexilMapping;
    private Map<String, Integer> m_moPlexilToPlanStructureMapping;

    public Any translateMessageContent(Any content, String from, String to)
        throws TranslationException {
        if (translates(from, to)) {
            if (from.equals(DecisionSupportAgent.LANGUAGE.toLowerCase())) {
                // PlanStructure to PLEXIL
                try {
                    ...
                    Any oTranslatedContent = ORBHelper.getORB().create_any();
                    oTranslatedContent.insert_string(sPlexil);
                    return oTranslatedContent;
                } catch (Throwable tx) {
                    throw new NotTranslatableException("Failed to translate message content.", tx);
                } // end try
            } else {
                // PLEXIL to PlanStructure
                ...
            } // end if
        } else {
            throw new TranslationNotSupportedException("Unable to translate message content from "+
                from+" to "+to);
        } // end if
    } // translateMessageContent
}
```

# Translation Service

## Writing a Translator (3/3)

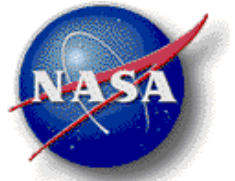


- Example (Cont'd)

```
public Any translateDataContent(IDataInfo info, Any content, String from, String to)
    throws TranslationException {
    if (translates(from, to)) {
        // only support translation from plexil to ExecStatus/PlanStructure
        if (from.equals(UniversalExecutive.LANGUAGE.toLowerCase())) {
            try {
                String sEvent = content.extract_string();
                ExecStatus oStatus = ExecutiveStatusConverter.convertXMLStatus(this, sEvent);
                if (oStatus != null) {
                    Any oTranslatedContent = ORBHelper.getORB().create_any();
                    ExecStatusHelper.insert(oTranslatedContent, oStatus);
                    return oTranslatedContent;
                } else {
                    throw new NotTranslatableException("Failed to translate data content.");
                } // end if
            } catch (Throwable tx) {
                throw new NotTranslatableException("Failed to translate data content.", tx);
            } // end try
        } else {
            throw new TranslationNotSupportedException("Unable to translate message content from "+
                from+" to "+to);
        } // end if
    } else {
        throw new TranslationNotSupportedException("Unable to translate message content from "+
            from+" to "+to);
    } // end if
} // translateMessageContent

} // PlanStructurePlexilTranslator
```

# Management Service

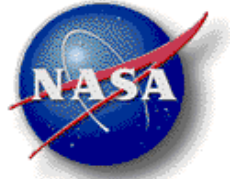


*Automation for Operations*

---

- Overview
- Heart Beat Publication
- Heart Beat Subscription
- Managed Attributes
- Managed Operations
- Operation Executors

# Management Service Overview

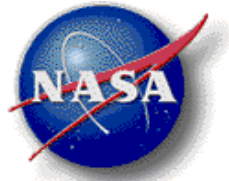


*Automation for Operations*

---

- Provides remote control and monitoring service for an actor
- Used to control and monitor actor's state
- Used to remotely configure an actor
- Management messages processed by management service not the actor
- CI Console uses the management service to monitor, control and configure actors, actor hosting environments and process managers

# Management Service Heart Beat Publication (1/2)



*Automation for Operations*

- Actor can have the MS periodically publish a heart beat
- Heart beat specified actor's current status
- Heart beat is automatically started if the interval is specified for an actor in its descriptor

```
<PROPERTIES>
```

```
  <PROPERTY name="ci.actor.heartbeat.interval">2500</PROPERTY>
```

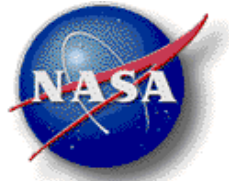
```
</PROPERTIES>
```

- MS used the DDS to publish the heart beat (domain: HeartBeat, topic: actor.qualifiedname)
- Actor can itself control the heart beat publication
- Method signatures

```
public void startHeartBeat(int interval) throws ServiceException;
```

```
public void stopHeartBeat() throws ServiceException;
```

# Management Service Heart Beat Publication (2/2)



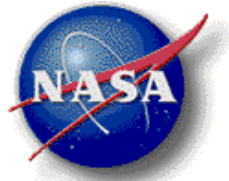
*Automation for Operations*

---

- Example use

```
try {  
    getManagementService().startHeartBeat(2500);  
    getManagementService().stopHeartBeat();  
} catch (Throwable tx) {  
} // end try
```

# Management Service Heart Beat Subscription (1/2)



## Automation for Operations

- The MS provides methods to register heart beat listeners used to monitor the status of another actor.
- Actor to be monitored must be identified by its qualified name

- Method signatures

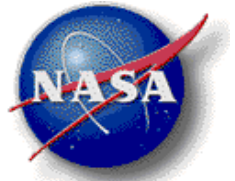
```
public void subscribeForHeartBeat(  
    String name,  
    IHeartBeatListener listener) throws ServiceException;
```

```
virtual void unsubscribeForHeartBeat(String name) throws ServiceException;
```

- Example use

```
try {  
    ExecutiveHBLListenerImpl oListener = new ExecutiveHBLListenerImpl();  
    IHeartBeatListener oListenerRef = oListener._this(ORBHelper.getORB());  
    getManagementService().subscribeForHeartBeat("gov.nasa.executive.PowerExecutive", oListenerRef);  
    getManagementService().unsubscribeForHeartBeat("gov.nasa.executive.PowerExecutive");  
} catch (Throwable tx) {  
} // end try
```

# Management Service Heart Beat Subscription (2/2)

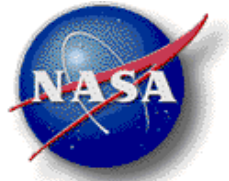


*Automation for Operations*

- Example use (Cont'd)

```
public class ExecutiveHBListenerImpl extends IHeartBeatListenerPOA {  
    public void onNewHeartBeat(  
        Credentials actor,  
        ActorStatus status) {  
        // process the status  
        if (status.Alive) {  
  
        } else {  
  
        } // end if  
    }  
} // ExecutiveHBListenerImpl
```

# Management Service Managed Attributes (1/3)



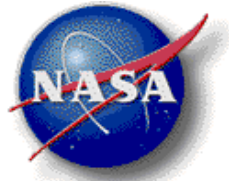
- Managed attributes are used to allow for the remote monitoring and optional control of an actor's configuration.

```
struct ManagedAttribute {  
    string Name;  
    string Type;           // IDL basic type or qualified user defined type  
    boolean IsReadOnly;  
} // ManagedAttribute
```

- Method signatures

```
public boolean addManagedAttribute(ManagedAttribute attr)  
    throws ServiceException  
public ManagedAttribute getManagedAttribute(String name)  
    throws NoSuchAttributeException, ServiceException  
public ManagedAttribute[] getManagedAttributes()  
    throws ServiceException  
public void setAttributeValue(String name, Any value)  
    throws NoSuchAttributeException, ServiceException  
public Any getAttributeValue(String name)  
    throws NoSuchAttributeException, NoValueException, ServiceException  
public boolean removeManagedAttribute(String name)  
    throws ServiceException  
public int addManagedAttributeChangeListener(  
    IManagedAttributeChangeListener listener)  
    throws ServiceException  
public void removeManagedAttributeChangeListener(int listenerID)  
    throws ServiceException
```

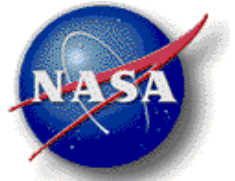
# Management Service Managed Attributes (2/3)



- Example use

```
try {
    // add a managed attribute
    ManagedAttribute oAttribute = new ManagedAttribute();
    oAttribute.Name = "enabled";
    oAttribute.Type = "boolean";
    oAttribute.IsReadOnly = false;
    getManagementService().addManagedAttribute(oAttribute);
    // set initial value
    Any oEnabled = ORBHelper.getORB().create_any();
    oEnabled.insert_boolean(true);
    getManagementService().setAttributeValue("enabled", oEnabled);
    // register a listener to be notified of any value changes made to the
    // attribute's value by a remote console/actor
    MyChangeListenerImpl oChangeListener = new MyChangeListenerImpl();
    IManagedAttributeChangeListener oChangeListenerRef =
        oChangeListener._this(ORBHelper.getORB());
    int nID =
        getManagementService().addManagedAttributeChangeListener(oChangeListenerRef);
    getManagementService().removeManagedAttributeChangeListener(nID);
    getManagementService().removeManagedAttribute("enabled");
} catch (Throwable tx) {}
```

# Management Service Managed Attributes (3/3)

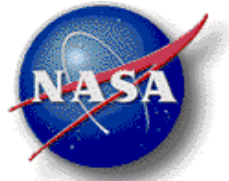


*Automation for Operations*

- Example use (Cont'd)

```
public class MyChangeListenerImpl extends IManagedAttributeChangeListenerPOA {  
    public void onAttributeChange(ManagedAttributeChangeEvent evt) {  
        // process the event  
        if (evt.Name.equals("enabled")) {  
            Credentials oSource = evt.Source;  
            Any oOldValue = evt.OldValue;  
            Any oNewValue = evt.NewValue;  
            boolean bEnabled = oNewValue.extract_boolean();  
            // enable or disable the actor  
        } // end if  
    } // onAttributeChange  
} // MyChangeListener_impl
```

# Management Service Managed Operations (1/3)



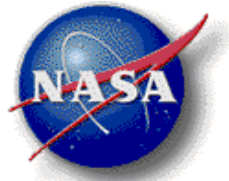
*Automation for Operations*

- Managed Operations are used to publish operations that can be invoked to control or configure the actor.
- Execution of managed operations is performed by the management service (not the actor) invoking the registered operation executor for the operation.

```
struct ManagedOperation {
    string Name;
    ManagedOperationParameterList Parameters;
    string ReturnType;          // IDL basic type or qualified user defined type or "void"
} // ManagedOperation
typedef sequence<ManagedOperationParameter> ManagedOperationParameterList;
struct ManagedOperationParameter {
    string Name;
    string Type;              // IDL basic type or qualified user defined type
} // ManagedOperationParameter
```

- Each actor has a fixed set of managed operations and operation executors registered by the CI: start, suspend, resume, stop, reset(CheckPoint), shutdown, getStatus, getState

# Management Service Managed Operations (2/3)



*Automation for Operations*

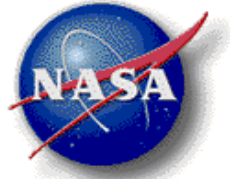
- **Method signatures**

```
public boolean addManagedOperation(ManagedOperation operation)
    throws ServiceException
public ManagedOperation getManagedOperation(String name)
    throws NoSuchOperationException, ServiceException
public ManagedOperation[] getManagedOperations()
    throws ServiceException
public boolean removeManagedOperation(String name)
    throws ServiceException
```

- **Example use**

```
try {
    IManagementService oMS = getManagementService();
    // create the operation
    ManagedOperation oOperation = new ManagedOperation();
    oOperation.Name = "reset";
    ManagedOperationParameter[] aoParameters = new ManagedOperationParameter[1];
    ManagedOperationParameter oParameter = new ManagedOperationParameter();
    oParameter.Name = "cp";
    oParameter.Type = "gov::nasa::ci::corba::api::CheckPoint";
    aoParameters[0] = oParameter;
    oOperation.ReturnType = "void";
}
```

# Management Service Managed Operations (3/3)



*Automation for Operations*

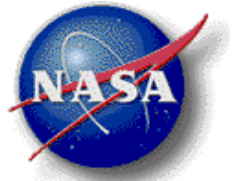
---

- Example use (Cont'd)

```
// register the operation
oMS.addManagedOperation(oOperation);

oMS.removeManagedOperation("reset");
} catch (Throwable tx) {}
```

# Management Service Operation Executors (1/3)

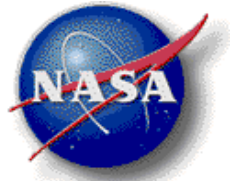


*Automation for Operations*

- Operation executors provide the implementation for the registered operations
- Operation executors are invoked by the management service when a request for the operation is received
- Method signatures

```
public void setOperationExecutor(  
    String name, IManagedOperationExecutor executor)  
    throws NoSuchOperationException, ServiceException  
public IManagedOperationExecutor getOperationExecutor(String name)  
    throws NoSuchOperationException, NoSuchExecutorException, ServiceException  
public void removeOperationExecutor(String name)  
    throws ServiceException
```

# Management Service Operation Executors (2/3)



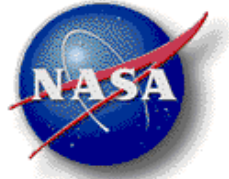
- Example use

```
try {
    IManagementService oMS = getManagementService();

    // create the executor
    ResetExecutorImpl oExecutor;
    IManagedOperationExecutor oExecutorRef =
        oExecutor._this(ORBHelper.getORB());
    oMS.setOperationExecutor("reset", oExecutorRef);

    oMS.removeOperationExecutor("reset");
} catch (Throwable tx) {}

public class ResetExecutorImpl extends IManagedOperationExecutorPOA {
```



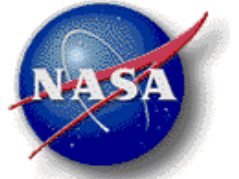
- Example use (Cont'd)

```
public void processOperation(
    String name,
    Property[] arguments,
    AnyHolder result) throws BadArgumentException, MissingArgumentException, InvocationException {
    // make sure it is the correct operation
    if (!name.equals("reset")) {
        Any oError = ORBHelper.getORB().create_any();
        oError.insert_string("Invalid operation name");
        throw new InvocationException("Invalid operation name",
            IManagementServiceErrorCodes.INVALID_OPERATION_ERROR,
            oError);
    } // end if

    // get the arguments for the operation
    CheckPoint oCP = null;
    Any oValue = PropertyListHelper::getPropertyValue(arguments, "cp");
    if (oValue != null) {
        try {
            oCP = CheckPointHelper.extract(oValue);
        } catch (Throwable tx) {
            throw new BadArgumentException("Argument value for parameter 'cp' is not of type CheckPoint.", "cp");
        } // end if
    } else {
        throw new MissingArgumentException("Missing required 'CheckPoint cp' argument", "cp");
    } // end if

    // invoke the operation
    getActor().reset(oCP);
} // processOperation
} // ResetExecutorImpl
```

# Time Service

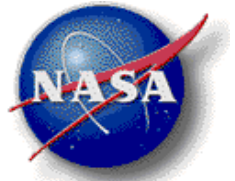


*Automation for Operations*

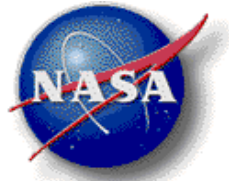
---

- Overview
- Time services

# Time Service Overview



- Provides time services as were provided by the CI in FY06 except for simulation time support
- One Time Service per actor hosting environment to prevent a single point of failure
- Time synchronization currently requires installation and configuration of NTP (Network Time Protocol) on all operating systems used for A4O



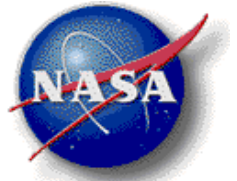
- **Method signatures**

```
public long getDateTimeMS();  
public DateTime getDateTime();  
public int notifyAtInterval(  
    ITimeServiceSubscriber subscriber,  
    int interval);  
public unsubscribe(int subscriberID);
```

- **Example use**

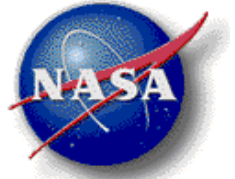
```
try {  
    IDataObject oDataObject = new DataObjectHelper();  
    oDataObject.setTimestamp(oTS.getDateTimeMS());  
} catch (Throwable tx) {}
```

# Logging Service

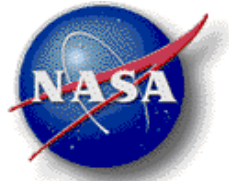


- Overview
- Logging Methods

# Logging Service Overview



- Process Manager and Actor Hosting Environment use Java's Log4J to log messages. Log4J is highly configurable using a configuration file (property or XML based) located in the same directory as the actor hosting environment/process manager's descriptor file with the same name followed by `Logger.cfg` or `Logger.xml`
- Logging Service logs messages using Log4J
- Logging Service generally only needed for C++ actors. C++ actors however can opt to use the ACE logging service since using the Logging Service is not efficient yet. Actor hosting environment's descriptor will have an option to specify the ACE `svc.cfg` file to use to configure the ACE logging service.



- Defining Logger instance per class

```
import org.apache.log4j.Logger;  
private final static Logger LOGGER = Logger.getLogger(<class>.class)
```

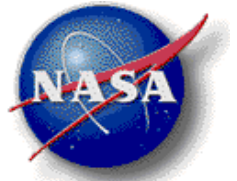
- Method signatures

```
public boolean isDebugEnabled();  
public void debug(Object message);  
public void debug(Object message, Throwable tx);  
public void info(Object message);  
public void info(Object message, Throwable tx);  
public void warn(Object name);  
public void warn(Object name, Throwable tx);  
public void error(Object name);  
public void error(Object name, Throwable tx);  
public void fatal(Object name);  
public void fatal(Object name, Throwable tx);
```

- Example use

```
public class PowerExecutiveActor {  
    private final static Logger LOGGER = Logger.getLogger(PowerExecutiveActor.class);  
  
    public void start() throws ActorException {  
        LOGGER.info("Starting actor...");  
    } // start  
} // PowerExecutiveActor
```

# Actor Hosting Environment

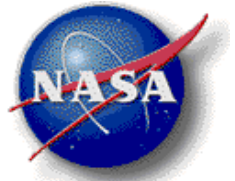


*Automation for Operations*

---

- Overview
- Configuration
- Actor Deployment

# Actor Hosting Environment Overview

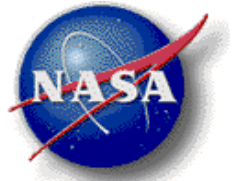


*Automation for Operations*

---

- Hosts one or more actors (C++ and/or Java)
- Provides core services to hosted actors
  - One shared directory service
  - One shared data distribution service
  - One shared translation service
  - One shared time service
  - One shared logging service
  - One transport service per actor
  - One data distribution service interface per actor
  - One management service per actor
- Manages the life cycle of an actor
- Is itself an actor

# Actor Hosting Environment Configuration (1/2)



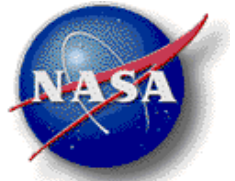
Automation for Operations

---

- Configured using a specialized actor descriptor (extension **.cihx**)

```
<HOSTING_ENVIRONMENT>  
</HOSTING_ENVIRONMENT>
```

- Specifies
  - Credentials
  - Advertisement
  - Transport (optional)
  - *Actor Service Provider*
  - *Actors*
  - *Properties*



- Hosting environment specific sections

```
<ASP>
```

```
  <DESCRIPTOR>heasp/ASP.ciaspx</DESCRIPTOR>
```

```
</DIRECTORY>
```

```
<ACTORS>
```

```
  <ACTOR>
```

```
    <DESCRIPTOR>PowerExecutive.ciax</DESCRIPTOR>
```

```
  </ACTOR>
```

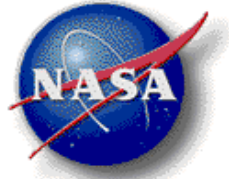
```
</ACTORS>
```

- One hosting environment specific property

```
<PROPERTY
```

```
  name="ci.ahe.deploydir">/usr/ahe/powerexec/deploy</PROPERTY>
```

# Actor Hosting Environment Actor Deployment

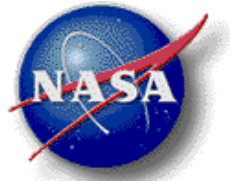


*Automation for Operations*

---

- Actor Service Provider descriptor must be located in the hosting environment's configuration directory or a directory relative to it.
- Actor descriptor files must be located in the hosting environment's deployment directory
- Hosting environment automatically loads, initializes and starts actors defined in the hosting environment's descriptor.
- Through the hosting environment management service it is possible to obtain a list of all actor's descriptors in the deployment directory. The hosting environment can be requested to load and initialize actors from the descriptors.
- Through the actor's management service actors can be started.
- Through the hosting environment management service actors can also be unloaded.

# Actor Service Provider

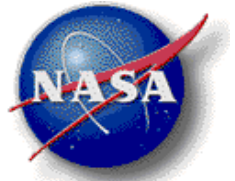


*Automation for Operations*

---

- Overview
- Configuration

# Actor Service Provider Overview

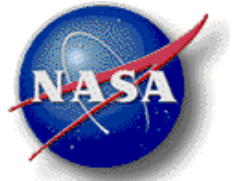


*Automation for Operations*

---

- Manages all CI services (transport, directory, data distribution, translation, time, logging)
- Manages actors
- Provides interface to CI services to registered actors

# Actor Service Provider Configuration (1/2)



*Automation for Operations*

---

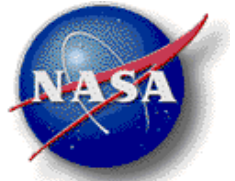
- Configured using a specialized actor descriptor (extension **.ciaspx**)

```
<ASP>
```

```
</ASP>
```

- Specifies
  - Transport Service (Required)
  - Directory Service (Optional)
  - Data Distribution Service (Optional)
  - Translation Service (Optional)
  - Properties

# Actor Service Provider Configuration (2/2)



*Automation for Operations*

- **Service descriptor references**

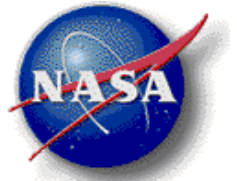
```
<TRANSPORT_SERVICE>  
  <DESCRIPTOR>TransportService.cicx</DESCRIPTOR>  
</TRANSPORT_SERVICE>  
<DIRECTORY_SERVICE>  
  <DESCRIPTOR>DirectoryService.cidx</DESCRIPTOR>  
</DIRECTORY_SERVICE>  
<DATA_DISTRIBUTION_SERVICE>  
  <DESCRIPTOR>DataDistributionService.cidx</DESCRIPTOR>  
</DATA_DISTRIBUTION_SERVICE>  
<TRANSLATION_SERVICE>  
  <DESCRIPTOR>TranslationService.cidx</DESCRIPTOR>  
</TRANSLATION_SERVICE>
```

- **One property**

```
<PROPERTY name="ci.asp.dir.config">.</PROPERTY>
```

Specifies the directory in which to find the service descriptor files. '.' indicates the same directory as the ASP's descriptor.

# Process Manager

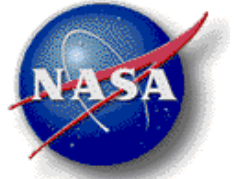


*Automation for Operations*

---

- Overview
- Configuration
- Process Descriptors

# Process Manager Overview

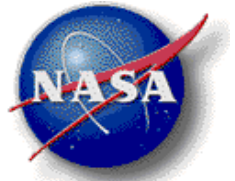


*Automation for Operations*

---

- One Process Manager per Operating System
- Preferably started as a service/daemon
- Used to start/stop/monitor actor hosting environment processes
- Processes are defined using process descriptors
- Is itself a special actor not requiring a hosting environment with built-in transport, data distribution and management services.

# Process Manager Configuration (1/2)



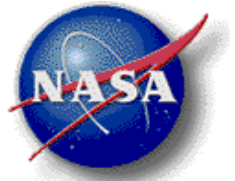
Automation for Operations

---

- Configured using a specialized actor descriptor (extension **.cimx**)

```
<PROCESS_MANAGER>  
</PROCESS_MANAGER>
```

- Specifies
  - Credentials
  - Advertisement
  - Transport (optional)
  - *Actor Service Provider*
  - *Processes*
  - *Properties*



- **Process manager specific sections**

```
<ASP>
```

```
  <DESCRIPTOR>pmasp/ASP.ciaspx</DESCRIPTOR>
```

```
</ASP>
```

```
<PROCESSES>
```

```
  <PROCESS>
```

```
    <DESCRIPTOR>HEPowerExecutiveProcess.cipx</DESCRIPTOR
```

```
  >
```

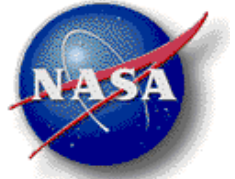
```
    </PROCESS>
```

```
  </PROCESSES>
```

- **One process manager specific**

# Process Manager

## Process Descriptor (1/4)



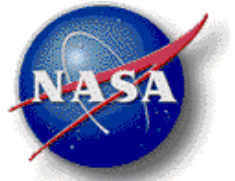
### Automation for Operations

---

- For each process that should be startable by the process manager a process descriptor must be defined and placed in the process manager's deployment directory.
- Process descriptor's file extension is **.cipx**  

```
<PROCESS id="gov.nasa.ci.HEPowerExecutive" name="HEPowerExecutive">  
</PROCESS>
```
- Specifies:
  - Process description
  - Environment variables
  - Program to start
  - Working directory
  - Optional Java specific settings for Java applications
    - boot classpath
    - extension classpath
    - classpath
    - VM options
    - Main class
  - Program arguments

# Process Manager Process Descriptor (2/4)



- Example

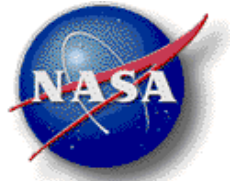
```
<PROCESS id="gov.nasa.ci.HEPowerExecutive" name="HEPowerExecutive">
  <DESCRIPTION>
    Starts the Actor Hosting Environment running the Power Universal Executive.
  </DESCRIPTION>

  <ENVIRONMENT>
    <SET name="JAVA_HOME">C:/JDK6</SET>
    <SET name="APPLICATION_ROOT">D:/CI/Testing/cidistribution</SET>
    <SET name="CI_AHE_ROOT">D:/CI/Java/cihostingenvironment</SET>
    <SET name="CI_INTERFACE_ROOT">D:/CI/Java/ciinterface</SET>
    <SET name="CI_EXAMPLES_ROOT">D:/CI/Java/ciexamples</SET>
    <SET name="JACORB_ROOT">D:/JacORB_2.3.0</SET>
    <SET name="LOG4J_ROOT">D:/Log4J/logging-log4j-1.2.14</SET>
  </ENVIRONMENT>

  <PROGRAM>
    $JAVA_HOME/bin/java.exe
  </PROGRAM>

  <WORKING_DIRECTORY>
    $APPLICATION_ROOT
  </WORKING_DIRECTORY>
```

# Process Manager Process Descriptor (3/4)



- Example (Cont'd)

```
<JAVA>
  <BOOTCLASSPATH>
    <PATHELEMENT>$JACORB_ROOT/lib/jacorb.jar</PATHELEMENT>
    <PATHELEMENT>$JACORB_ROOT/lib/logkit-1.2.jar</PATHELEMENT>
    <PATHELEMENT>$JACORB_ROOT/lib/avalon-framework-4.1.5.jar</PATHELEMENT>
  </BOOTCLASSPATH>

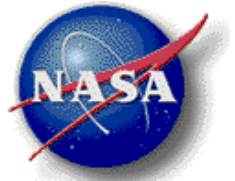
  <EXTENSIONCLASSPATH>
    <PATHELEMENT>$CI_INTERFACE_ROOT/dist</PATHELEMENT>
    <PATHELEMENT>$CI_AHE_ROOT/dist</PATHELEMENT>
    <PATHELEMENT>$CI_EXAMPLES_ROOT/dist</PATHELEMENT>
    <PATHELEMENT>$LOG4J_ROOT/dist/lib</PATHELEMENT>
  </EXTENSIONCLASSPATH>

  <CLASSPATH>
    <PATHELEMENT>$APPLICATION_ROOT/config</PATHELEMENT>
  </CLASSPATH>

  <VMOPTIONS>
    <VMOPTION>-Xincgc</VMOPTION>
    <VMOPTION>-Xmx768m</VMOPTION>
    <VMOPTION>-Xss1024k</VMOPTION>
    <VMOPTION>-Xms32m</VMOPTION>
  </VMOPTIONS>
```

# Process Manager

## Process Descriptor (4/4)



*Automation for Operations*

- Example (Cont'd)

```
<MAINCLASS>gov.nasa.ci.ahe.HostingEnvironment</MAINCLASS>  
  
</JAVA>  
  
<ARGUMENTS>  
  <ARGUMENT>HEPowerExecutive</ARGUMENT>  
</ARGUMENTS>  
</PROCESS>
```

- The CI Console can obtain the list of all available process descriptors for a process manager and select which processes to load and start.
- The CI Console can monitor the status of all processes and if necessary stop and unload processes.