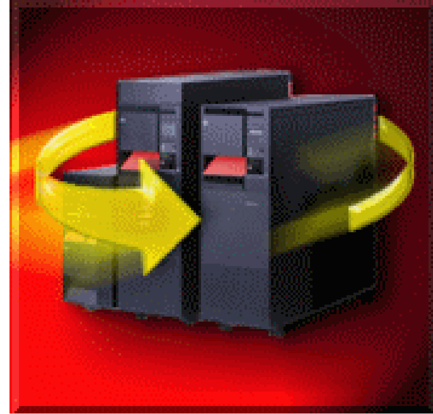
How To...



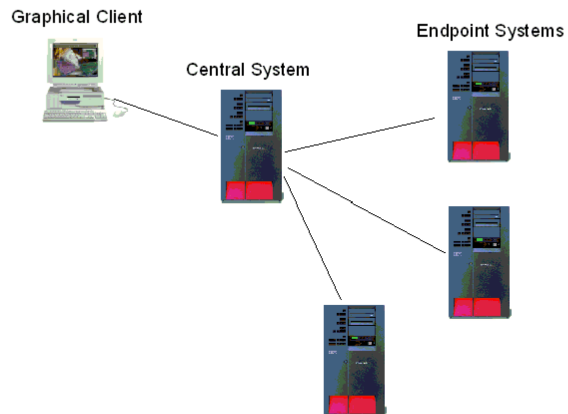
Distributing Command and API calls using the new Management Central Java Framework

VeeFiveAreOneEmZero
Final Version

Document last changed: June 27, 2001 11:25am

Preface

"Managing multiple systems as easy as managing a single system."



Management Central Documentation

This *Management Central How To...* document is a bridge between the code you need to write for your application and the JavaDoc provided by the Management Central Java Framework. To assist in your application development, you should first find the documentation listed below and take a quick look at it. Next, read this document to get a high level understanding of what is available for your application development with tips on design and implementation. When you are finished reading this book, refer to the JavaDoc for the details about the classes and interfaces discussed in this book that will apply to your application.

Management Central Web Site:

http://www.as400.ibm.com/sftsol/MgmtCentral.htm

Java Class Documentation (JavaDoc):

AS/400 Toolbox for Java:

http://www-1.ibm.com/servers/eserver/iseries/toolbox/

What This How To Book Contains

This book is organized to provide information for the GUI developer who wishes to make use of the Management Central Distributed Command Call and Distributed API Call interfaces. These interfaces will allow a single AS/400 CL Command or API to be executed on any number of endpoint systems.

This discussion represents a very small functional subset of the Management Central Java Framework, and since only client-side activities are necessary to accomplish a distributed Command or API call, all server activities will be abstracted from this discussion. If your intent is to develop your own server side application using the Management Central Java Framework as a base for your application, a more inclusive version of this document, called "How To... Implementing to the new Management Central Java Framework" should be consulted as a reference. This document can be found in the *Unity Java Institute* at http://w3.rchland.ibm.com/projects/AS400-Unity/UnityInst/UICourses.html.

Each major section or chapter contains the follow information:

- Overview - a brief description of the topic being discussed
- Interfaces and Flows - Steps necessary for the GUI Developer.
- Terms, Classes, and Interfaces - a table of Java terms, classes, and interfaces that are used
- Scenarios - commonly used development scenarios with code snippets
- Hints and Tips - technique hints and tips when designing and developing your application

Conventions Used in this Cookbook

This book uses the following typographical conventions:

This style....	Is used for...
`Fixed width font`	Code elements such as classes and methods.
`Fixed width font underline`	Emphasis for code elements.
Bold	Management Central Classes and Interfaces.
Underlined	Management Central Methods.

Table of Contents

INTRODUCTION 1
MANAGEMENT CENTRAL DISTRIBUTED TASKS 3
OVERVIEW 3
  GUI Developer 4
KNOW YOUR MANAGEMENT CENTRAL OBJECTS 5
MANAGEMENT CENTRAL DISTRIBUTED COMMAND CALL APPLICATION 8
OVERVIEW 8
INTERFACES AND FLOWS 8
  Application Designer 8
  GUI Developer 9
  Scenario 1: Create and Execute a Distributed Command Call Task 9
  Scenario 2: Get list of Distributed Command Call Tasks 10
  Scenario 3: Delete a Distributed Command Call Task 10
  Scenario 4: Change a Distributed Command Call Task 11
  Scenario 5: Get asynchronous status and results of a Task 11
TERMS, CLASSES, AND INTERFACES 13
HINTS AND TIPS 14
PROGRAMMING EXAMPLES 15
MANAGEMENT CENTRAL DISTRIBUTED API APPLICATION 17
OVERVIEW 17
INTERFACES AND FLOWS 17
  Application Designer 17
  Gui Developer 18
  Scenario 1: Create and Execute a Distributed API Application Task 18
  Scenario 2: Get a list of Distributed API Application Tasks 20
  Scenario 3: Delete a Distributed API Task 20
  Scenario 4: Change a Distributed API Task 21
  Scenario 5: Get asynchronous status and results of a Task 21
TERMS, CLASSES, AND INTERFACES 23
PROGRAMMING EXAMPLES 25
ADVANCED FEATURES 27
SAVING ENDPOINT SYSTEMS AND SYSTEM GROUPS 27
  Scenario 1: Creating Definition Instances 27
  Scenario 2: Retrieve your Definitions 28
  Scenario 3: Deleting Definition Instances 28
  Scenario 4: Changing an existing Definition Instance 29
GET ASYNCHRONOUS STATUS UPDATES FOR LIST OF TASKS 29
PRIVATE DESCRIPTORS 30

Public Descriptor Sharing 31
Auto Increment 32
Categories 32
Schedule Your Task 33
Get list of Scheduled Distributed Tasks 34
Handling Exceptions 35
Tracing Messages 36
Additional Utilities 38

Introduction

If you are new to Management Central, here are a some basic principles, behaviors, and a brief history of Management Central.

Principles of Management Central:

- Make the management of multiple systems as easy as managing a single system
- Provide this management capability in the base AS/400 operating system
- Provide an easy-to-use graphical user interface to management functions

Management Central Application behaviors:

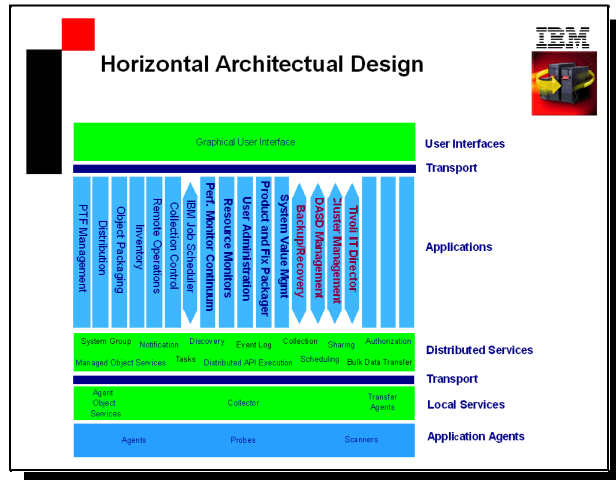
- Live GUI updates (automatic refresh)
- Attended or unattended
- Immediate or scheduled
- Multiple System
- Short or Long running

A little bit of history...
Management Central is a suite of integrated systems management applications that began to appear in V4R3 with Client Access for Windows (5763-XD1) V3R2. When installing Client Access for Windows, you can select to perform a Custom installation and optionally choose to install Management Central along with Operations Navigator.

With V4R3, Management Central provided the base for multiple system management with the introduction of the Management Central C++ infrastructure. This infrastructure provided a horizontal architectural approach to software development when developing AS/400 system management solutions. This horizontal approach separates the user interface from the transport mechanism, the application from common service components, etc. AS/400 Endpoint Systems, AS/400 System Groups, Event Log, and a Monitor application provided an intuitive graphical interface to real-time performance information with simple automation and notification for management of multiple AS/400 systems.

In V4R4, Management Central added a number of new integrated graphical applications to help manage AS/400 systems: Inventory Collection, Software Fix (PTF) Management, Remote Operations, Package and Object Distribution, and Performance Collection Services. This extended the Management Central C++ horizontal infrastructure with additional common services like Bulk Data Transfer, Discovery, and Collection Services. Management Central is now an integrated part of AS/400 Operations Navigator in V4R4. The Operations Navigator tree hierarchy has been enhanced to include Task Activity, Scheduled Tasks, Definitions, Monitors, AS/400 Endpoint Systems, and AS/400 System Groups.

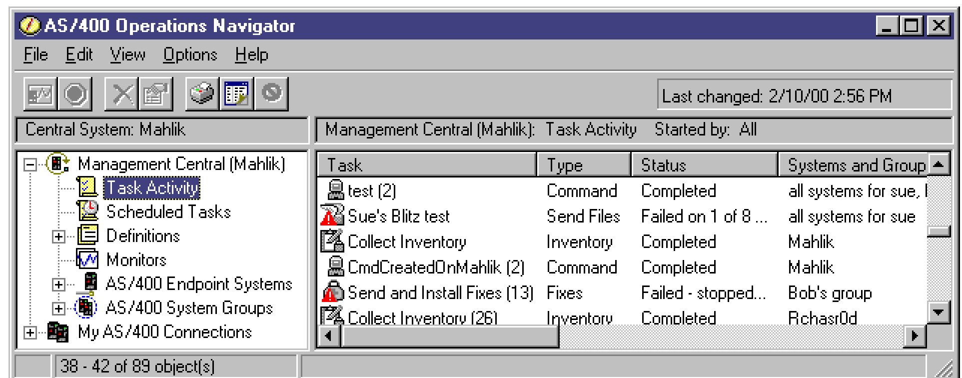
In V5R1, Management Central continues to extend its AS/400 management control with additional applications like enhancing historical Monitor capabilities, Product and Fix Packager, Job and Message Resource Monitors, User Profile Management, and System Value Management. With all the interest in Management Central, areas within IBM and external to IBM are looking at using Management Central to implement their management applications and solutions. For this reason, we have also developed the Management Central Java Framework(jMC).



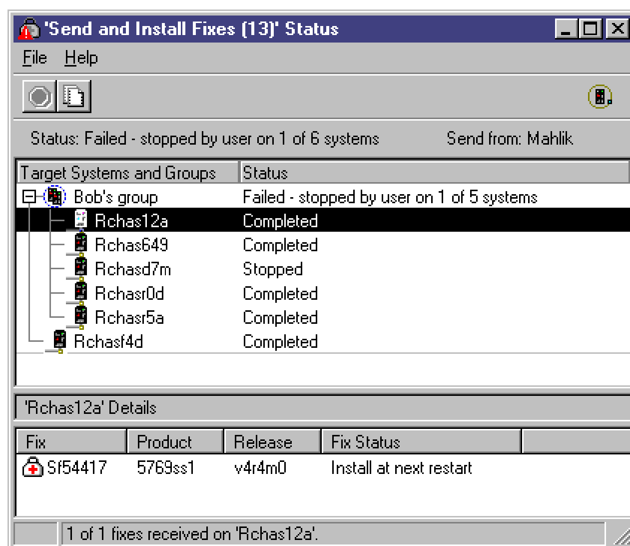
The Management Central Java Framework is an extendible and pluggable infrastructure for areas within IBM and external system management solution partners to use to their advantage when developing their suite of applications. Areas within IBM such as DASD Management, Backup Recovery and Media Services(BRMS), Clusters, and LPAR are using the jMC to build functions that balance disk drives, backup and restore data, and build system groups based on hardware configurations.

Management Central Distributed Tasks

Overview



Tasks are long running asynchronous operations that can be scheduled and run unattended on multiple remote systems. The operator or administrator generally selects an action to perform from the graphical user interface, selects systems on which the actions should be performed, and then determines whether to run the action now or schedule it to be run at a later date and time. After the action completes at the endpoint system, it sends status information back to the central system and can be later viewed on the workstation.



One example of a task is the Fixes application that was delivered with Management Central in V4R4. One function of this application allows the operator to distribute and install Program Temporary Fixes(PTFs) on other systems in the network. The operator would select a list of fixes to install and then pick which system groups or systems to install them on. During the distribution and installation process, status information would be sent back to the central system. If and when the operator wanted to check on the task, she could simply click on the Task Activity container and view the detailed status information for that task. Tasks will always have some final status whether it completed successfully, failed, or was ended by the user.

Currently the Management Central Java Framework provides three different ways for applications to implement tasks. The first method, and simplest, is to use the Distributed Command Call Application provided with Management Central. The Distributed Command Call Application allows you to execute an AS/400 CL command on a group of systems. If all or part of your application can be performed simply by sending a command to the remote system all you need to do is use the **McDistCommandDescriptor** class. This class takes a CommandCall object which you construct from the AS/400 Java Toolbox and a **McSystemGroup** to indicate where to execute the AS/400 CL

command. For more details see section Management Central Distributed Command Call Application on page 8.

If instead of calling a command you need to call an AS/400 program or service program, you can use the second way to implement tasks by using the Distributed API Application. By constructing Program Call Markup Language(PCML) statements and using the ProgramCallDocument from the AS/400 Java Toolbox, you can create tasks using the **McDistApiDescriptor** class. The **McDistApiDescriptor** class also accepts a ProgramCall class or ServiceProgramCall class from the AS/400 Java Toolbox. For more details see section Management Central Distributed API Application on page 17.

If neither of the provided applications meet your needs, then you can create a new type of task by extending the Management Central Java Framework. However, this scenario will not be covered in this document. If you've determined that simply using the functionality supplied by the Distributed Command and Distributed API Call interfaces does *not* meet your needs, then you should view the full version of this How-To document. This document can be found in the *Unity Java Institute* at http://w3.rchland.ibm.com/projects/AS400-Unity/UnityInst/UICourses.html. The full version of the document will guide you in developing your own server-side applications, and provide instructions on how to distribute your task to multiple endpoint systems using a graphical user interface.

GUI Developer

Other than providing aesthetics, the GUI developer has two main tasks in implementing the user interface for the application. The first is to connect the user interface to the Distributed Task Descriptor class created by the application developer. When using the Distributed Command Call or Distributed API Call applications provided by the Management Central Java Framework, the Descriptor class will be **McDistCommandDescriptor** or **McDistApiDescriptor**, respectively.

The second task is to decide where the user interface will reside. One possible solution could be to add a new task container in Operations Navigator under the Task Activity branch of the Management Central tree, adding a context menu option on this new container to create new tasks, and adding context menu choices on each task to view it's properties and to perform actions (e.g."New Based On", "View Status", etc.).

To perform this development you will need to know how to code your user interface to interact with the Management Central Java Framework **McDistributedTaskDescriptor**, **McDistributedTaskView**, and **McDistributedTaskListView** classes, how to use the GUI helpers provided by the Management Central Java Framework to display properties, select systems and groups to run, and delete your application tasks, and also how to create an Operations Navigator Plugin using **ListManager** and **ActionManager** interfaces. All this (and more!) will be covered in remainder of this document.

Know Your Management Central Objects

Within the Management Central Java Framework there are several categories of objects you'll need to become familiar with in order to use either of the distributed applications provided by the jMC. Some classes from each category were brushed upon in the preceding section; they and others will be classified and described here.

There are three main categories to be concerned with when dealing with the jMC. The first, classified as the **Action**, is the real meat of the application. It's the server side code that provides the activities and actions that define the application. It resides on the endpoint system, and is the workhorse that performs the application's goals on each individual system. For instance, in the Distributed Command Call application supplied by the jMC, **McEndpCommandAction** is the class that actually provides the mechanism to execute or cancel the command on each endpoint system.

For the GUI programmer, the most important category of Objects in the jMC is the **View**. Views provide a bridge between the graphical client and the server-side application. By creating and maintaining a reference to a View, many of the complexities involved with maintaining a remote reference to the server are abstracted from the GUI programmer. For instance, the View handles all connection details with the CentralSystem. The AS/400 that the user has specified as the central system is stored within Operations Navigator, and upon construction of any View object, this data is retrieved, and used to connect to the system. The interfaces that exist on the View objects are there to propagate data from the client to the server, and to maintain the integrity of that data; that is, when data is changed on the client, that change must be sent to the server to ensure the endpoint action is executed correctly.

Views come in a multitude of flavors, but at the topmost level resides the interface all Views must implement, **McManageableViewIfc**. Key methods from this interface are:

addManageable()	Allows the jMC to manage the data associated with this View on the Central Site. "Management" consists of (among other things) caching the data into memory, persistently storing it in a database, updating created and changed dates, and distributing the data to the endpoint systems when instructed to do so.
changeManageable()	Updates the state of a presently managed object. If, after calling addManageable(), the data stored within the View is changed, the jMC on the Central Site must be updated with the changed data. This method provides an interface for the update.
getManageable()	Allows for remote retrieval of managed objects from the Central Site into the View object. Techniques for selecting which managed objects should be returned will be discussed in depth in a later section.
removeManageable()	Once a managed object is no longer needed (because the associated Actions have completed, for instance), this method must be called on the

	object to allow the jMc to clean up any data associated with the managed object.

In addition, class **McManageableListView** exists to manage a list of Views. It allows the user to retrieve an entire list of qualifying View objects with a single call to the Server. Each View in the list can then be managed independently or as part of the ListView. Key methods from McManageableListView are:

getManageableViews()	Returns a list of qualifying View objects from the server.
removeManageableList()	Removes manageablility of each View that's part of this list from the server. The jMC will no longer manage any of the elements.

It's important to note that none of the specific details behind View objects need be known by the GUI developer. That is, you don't need to know how to manage objects within the Management Central Java Framework; rather, you simply need to tell the Framework which objects to manage.

The third crucial category of objects in the jMc is the actual managed object. A managed object is technically any object implementing the McManageable interface, but more specifically, a category of objects called **Descriptors**, provided by the jMC, has already extended the McManageable interface. The Descriptor contains data about the activity, and the action specific to the activity. This descriptor must be created by the graphical client, usually by gathering information from the user, and then placed within a View object. When addManageable() is called on the View, the Descriptor is passed along to the Central System, and, based on data within the Descriptor, the Action can then be performed.

An important detail that is abstracted from the developer is the distributed nature of activities within the jMC. When the descriptor is created on the client, a system group is defined within it. Then, when the View object is created, it sends the descriptor to the central system (the central system is retrieved from Operations Navigator - yet another detail abstracted from the user) where information about the activity can be managed. When you tell your task to exeute, the jMC handles all communication details to distribute and start the activity on the endpoint systems. Similarly, the endpoints maintain a reference to central system, and the central system maintains a reference to the client, and therefore can propagate status and results about the endpoint activity back to the client. (See the figure on the following page for details).

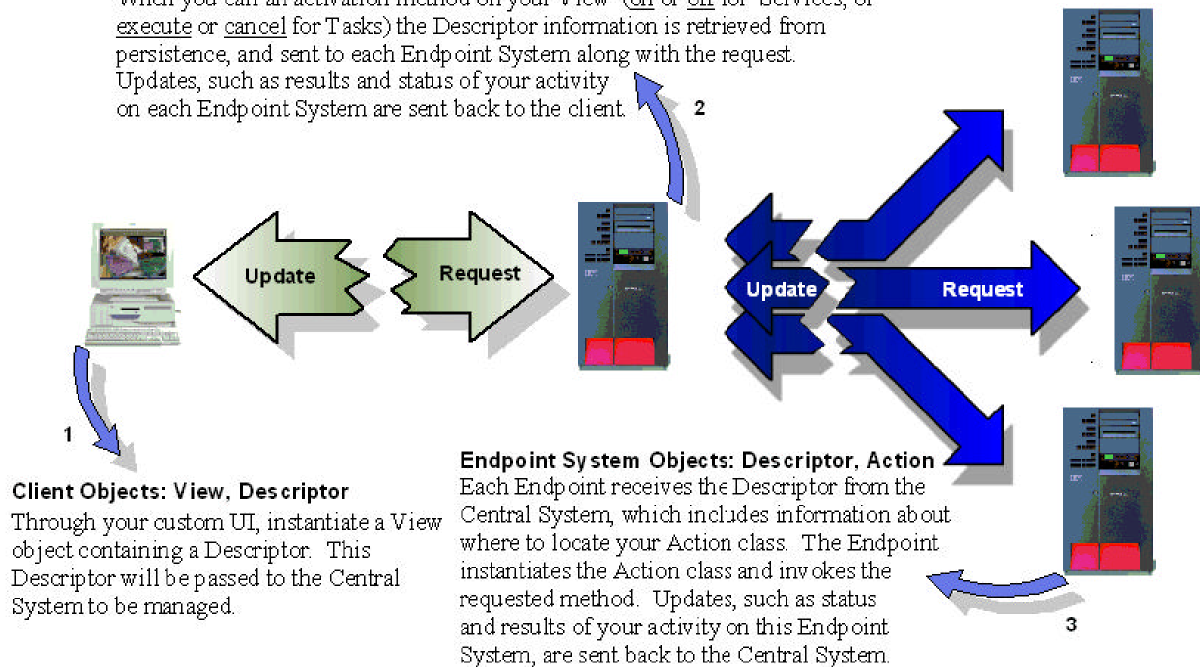
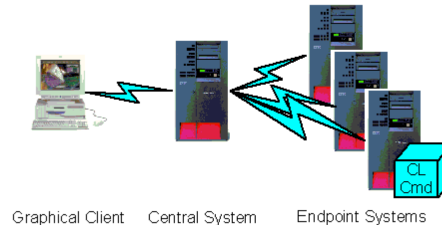


Figure: Objects and actions within the Management Central Java Framework.
When you call methods on your View object, the jMC interacts with the Central System to fulfill the request. If the request applies to the Endpoint Systems, as it would if, for instance, you asked to start a service, the Central System sends requests to each Endpoint System on your behalf. Updates from each request flow back to the client. While the Descriptor flows from client to Central System to Endpoint Systems, other Objects have their appointed place.

Management Central Distributed Command Call Application

Overview



In this chapter you will learn about classes provided by the Management Central Java Framework that allow you to run an AS/400 CL Command on multiple remote AS/400 endpoint systems. The **McDistCommandDescriptor** class, with the help from the **McDistributedTaskView** and **McDistributedTaskListView** classes, provide the Management Central Distributed Task functions allowing a Java program to execute a non-interactive AS/400 command on multiple systems and return status and results back to the Central System and the graphical client.

To make this happen, the jMC uses classes from the AS/400 Java Toolbox. The CommandCall class is used to construct the AS/400 CL command so that the jMC can send and execute the request to the endpoint systems. The CommandCall class is also used to store any AS400Message objects that are returned as the result of executing the command.

Interfaces and Flows

Application Designer

You will first want to determine whether the **McDistCommandDescriptor** class contains the functionality that meets your application needs. Your application would use this function if the interfaces you are calling on the AS/400 are CL commands and you require only minimal status and results about the execution of the command.

Some of the specifications of the **McDistCommandDescriptor** are:

- Can run a single AS/400 CL command simulatneously on multiple systems
- Runs asynchronously, meaning a once the task is distributed to the endpoints, each endpoint runs the task in parallel and reports status back upon completion of the task
- Returns a limited defined set of status values
- Returns AS/400 messages within the CommandCall object
- Command will run under the profile of the owner of the activity

GUI Developer

The following scenarios describe how to use the CommandCall, **McDistCommandDescriptor**, **McDistributedTaskView**, and **McDistributedTaskListView** classes.

Classes and Interfaces:

- com.ibm.mc.client.activity.task.command.McDistCommandDescriptor
- com.ibm.mc.client.activity.task.McDistributedTaskView
- com.ibm.mc.client.activity.task.McDistributedTaskListView
- com.ibm.mc.client.activity.McActivityDescriptorSelectionCriteria
- com.ibm.mc.client.McManageableSelectionCriteria
- com.ibm.as400.access.CommandCall

Scenario 1: Create and Execute a Distributed Command Call Task

It is very easy for the GUI developer to use the **McDistCommandDescriptor** class to create and execute a distributed command call task. Here are the steps to get you started:

1. Create an instance of an AS/400 Java Toolbox CommandCall object and set the command.
2. Create an instance of a Distributed Command Descriptor specifying the Task Name, Task Owner, Task Description, Sharing, and a System Group; then set the command using the CommandCall object created in step 1. Note that the getSystemGroup method used in the Descriptor's constructor must be supplied by the user to retrieve the list of endpoint systems on which to execute the command.
3. Create an instance of a Distributed Task View object specifying the Distributed Command Descriptor created in step 2.
4. Tell the Distributed Task View to add the instance of your task so that it can be managed.
5. Call the execute method to distribute and execute the command on all the endpoint systems specified in the System Group.

```
    McDistCommandDescriptor distCommandDesc = null;
    McDistributedTaskView   distCommandView = null;

    // Step 1: Create an instance of a CommandCall object and set the command
    CommandCall cmdToRun = new CommandCall();
    cmdToRun.setCommand("CRTLIB USRLIB");

    // Step 2: Create an instance of a Distributed Command Descriptor
    distCommandDesc = new
                   McDistCommandDescriptor("Command Task Name",          // Name
                                           "Command Task Description", //
Description
                                           McManageable.NONE,          // Sharing
                                           getSystemGroup(),           // System
Group
                                           cmdToRun,                   //
CommandCall
                                           null);                      //
Category
```

Hints and Tips: If you look at the Toolbox documentation for CommandCall, you will see a constructor that accepts an AS400 Object. In the example above, if you supply a CommandCall containing an AS400 Object that is already connected to some AS/400 endpoint system, the jMC will accept it, but will overwrite the Object. Since the AS400 is used to execute native calls on the Endpoint, the data stored within the Object must correspond with the current system. If it does not, the jMC will construct a new AS400 on the endpoint system and use that Object for command processing.

Scenario 2: Get list of Distributed Command Call Tasks

If you need to retrieve a list of Distributed Command Call tasks that would include the task you created in Scenario 1, this next step will show you how. There are only a few steps needed here.

1. Set up the selection criteria to only get tasks of the class **McDistCommandDescriptor**
2. Create an instance of the **McDistributedTaskListView** to manage the list of tasks
3. Ask the Distributed Task Manager to return to you a list of Distributed Command Call Tasks

```
    McManageableSelectionCriteria selCriteria    = null;
    Vector                        retrievedTasks = new Vector();

    // Step 1: Define selection criteria to get a list of Distributed Command Tasks
    selCriteria = new McManageableSelectionCriteria(
               "com.ibm.mc.client.activity.task.command.McDistCommandDescriptor", //
Class
               McManageable.ALL,  // Category
               null,              // List of owners (only used if next parameter is
true)
               false,             // Include shared activities
               0);                // Last changed date of the activity
```

Scenario 3: Delete a Distributed Command Call Task

If you need to delete a Distributed Command Call task, or a list of them, the Distributed Task View and List View classes provide the methods for you. Deleting a task removes the task from the Management Central databases and is no longer a managed task. When working with the View object, you can simply call removeManageable on the instance of the object itself; for a List of Views, you can simply call removeManageableList on the ListView instance.

```
    // Step 1: Tell the Distributed Task View to remove the instance of your task
    view.removeManageable();

    // Or, for a Distributed Task List View
```

Scenario 4: Change a Distributed Command Call Task

To save changes of an existing task on the Central System, the Distributed Task View provides the method for you to use. Prior to calling change, you would have a reference to a task that you previously created or retrieved from a list of tasks. With the reference to the task, you may have the end user modify it by displaying a property page and using the appropriate set methods to update the task instance. When you have the task instance up to date, you can tell the Distributed Task View to store the changes. When working with the task view object you can simply call changeManageable on the instance of the object itself.

```
    // Step 1: Tell the Distributed Task View to change the instance of your task
    view.changeManageable();
```

Scenario 5: Get asynchronous status and results of a Task

Once you've created your task and called execute, the task will run asynchronously with other activities. If you want to monitor the status of the request, you will want to attach a class to handle status and results that are returned from each endpoint system. This class will implement the **McStatusDetailListener** and **McResultDetailListener** interfaces. Note: In the example shown below, the same class implements both these interfaces so we pass in this as the object to handle status and result updates.

```
 // Previously you would have retrieved a list of task views and selected the
 // one task view that you want to monitor status and results

 // Attach to be notified when a Status or Result object is received
 view.attachStatusDetailListener(this);
 view.attachResultDetailListener(this);

 // Display a status window or dialog

 // When the user is done with this window or dialog, detach the status and
 // result listeners before closing the window



// Implementing the McStatusDetailListener interface
public void statusUpdate(McStatusEvent event) throws McException
{
  // Get the status object out of the event information
  McStatusIfc status = event.getStatus();

  // If the overall status value indicates the task has finished
  if ( status.getLevel() == McStatusIfc.DistributedAct && status.isFinalized()
)
  {
```

Hints and Tips: The statusUpdate method will be called on a separate thread when status is returned back to the listener. If you want your task to appear to run synchronously, then you can use the suspend method after performing the execute action and place a resume method call here in the statusUpdate method. Be aware, however, that since results are also asynchronous, you have no guarantee that results will be received before status.

```
// Implementing the McResultDetailListener interface
public void resultUpdate(McResultEvent event) throws McException
{
  // Get the result object out of the event information
  McResultIfc result = event.getResult();

  // Since the result object is a hierarchy of results for each Endpoint
System
  // specified in the task, you need to get the results for the specific
Endpoint
  // System to see its details.
  McResultIfc childResult = (McResultIfc)(result.findChild("system1"));

  // Be sure that the result object is an instance of CommandCall before
  // performing CommandCall type methods.
  if(childResult != null && childResult.getResultData() instanceof
CommandCall)
  {
    // Get the CommandCall object our of the result object
    CommandCall cmd = (CommandCall)childResult.getResultData();

    if ( (cmd.getMessageList() != null) && (cmd.getMessageList().length > 0) )
    {
      // Retrieve list of AS/400 messages
```

Terms, Classes, and Interfaces

Thing	Type	What to do with it	Purpose
McDistCommandDescriptor	Class	Use	An application will create one of these objects to execute an AS/400 CL command on multiple systems.
McDistributedTaskView	Class	Use	This class bridges your application to MC Java Framework functions to manipulate tasks and provide additional capabilities that make GUI programming easier. By using the attach and detach capabilities, GUI classes can be notified directly when a status has changed or when results have been received. Task Actions: - execute - displayScheduleDialog - schedule - cancel Distributed Task Manager: - addManageable - getManageable - changeManageable - removeManageable Event Listeners: - attachConnectionListener - detachConnectionListener - attachStatusDetailListener - detachStatusDetailListener - attachResultDetailListener - detachResultDetailListener
McDistributedTaskListView	Class	Use	This class bridges your application to MC Java Framework functions to manipulate lists of tasks and provide additional capabilities that make GUI programming easier. By using the attach and detach capabilities, GUI classes can be notified directly when a task has been created, changed, updated, and deleted. Distributed Task Manager: - getManageableViews - removeManageableList Event Listeners: - attachManageableListener - detachManageableListener
McManageableSelectionCriteria	Class	Use	Use this class in conjunction with the McDistributedTaskListView to specify the

			type of tasks to retrieve. The selection criteria allows you to specify Type, Category, Sharing, etc.
McActivityDescriptorSelectionCriteria	Class	Use	Like McManageableSelectionCriteria, you use this class in conjunction with the McDistributedTaskListView to specify the type of tasks to retrieve. This class extends the base to include selection criteria to subset activities based on their status values.
CommandCall	Class	Use	Provided by the AS/400 Java Toolbox: Contains the AS/400 CL command to execute on all the remote systems. This object will also be used to return AS/400 messages if the execution of the command resulted in any joblog messages.

Hints and Tips

What exactly does the execute do for a Distributed Command Call?

The execute tells the Management Central Java Framework to distribute the task to every system specified in the system group. Once delivered to the endpoint system, the CommandCall object will be extracted from the task and run. If the return code value from the run() method indicates an error (false), then McStatusIfc.Failed will be returned in the status event. If the return code value indicates success (true), then a value of McStatusIfc.Completed will be returned in the status event.

In either case, results are also constructed and returned to the Central System and available to the client. After the CommandCall object is run, any messages are placed in the CommandCall object. In your resultUpdate method, you can interrogate the result information, extract the CommandCall, and check to see if there are any messages that have been returned.

The actual execution of the command will occur in a Client Access Server job. This job will run under the user profile of the owner of the task. For more details, see the JavaDoc for CommandCall.

What exactly does the cancel do for a Distributed Command Call?

When the CommandCall object is requested to run on the endpoint system, it will start a new Client Access server job. The Distributed Command Call application will remember this job name. When the cancel request is received on the endpoint system, an ENDJOB immediate command will be executed to end the Client Access server job processing the execute request. If the ENDJOB command was executed a McStatusIfc.Canceled status will be returned in the status event. If the execute request had already completed, then the cancel request will be disregarded.

Programming Examples

In CMVC there are a number of test programs available at:
as400a\v5r1m0t.ss03\int\cmvc\java.pgm\yps.ss03\com\ibm\app\client

- TestCmdCall
- TestCmdCallAttach
- TestCmdCallCancel
- TestCmdCallCreate
- TestCmdCallExecute
- TestCmdCallRemove
- TestCmdCallSchedule

The **TestCmdCall** Java program is an all inclusive test program that will create a new task, attach for status and result notifications, execute the task, process status and result events, and remove the task when completed.

The rest of the test programs break the entire test into controllable pieces. The **TestCmdCallCreate** Java program will create a new task. The **TestCmdCallAttach** Java program will associate itself to the task so when the task executes it can receive status and result notifications. The **TestCmdCallExecute** Java program will kick off the execution of the task and will also receive status and result notifications. The **TestCmdCallSchedule** Java program will schedule the task to execute at a later date and time. The **TestCmdCallRemove** Java program will delete the task from the Management Central Task data base on the AS/400. The **TestCmdCallCancel** Java program will attempt to cancel the running task. The **TestCmdCallChange** Java program will change the name of the task.

In these examples it is important to understand the **McKey** concept. When a new task is created, a key is created to uniquely identify the task This key is made up of three parts: the task class, the task name, and the user who owns the task. In the **TestCmdCallCreate** Java program the key is created and assigned when you create a new instance of your task. This happens when you instantiate a new **McDistCommandDescriptor** and perform an addManageable.

```
    // Create an instance of a Distributed Command Descriptor
    McDistCommandDescriptor distCommandDesc = new
                   McDistCommandDescriptor("Command Task Name",         // Name
                                           "Command Task Description", //
Description
                                           McManageable.NONE,          //
Sharing
                                           getSystemGroup(),           // System
Group
                                           cmdToRun,                   //
CommandCall
                                           null);                      //
Category
```

Notice that when you construct a new **McDistCommandDescriptor**, you specified two out of the three essential parts of the key:

1. Task Class = McDistApiDescriptor
2. Task Name = "DistApiDesc-TestTask

The owner is determined by the Management Central Java Framework.

Now in the **TestCmdCallAttach**, **TestCmdCallExecute**, **TestCmdCallCancel**, **TestCmdCallChange**, and **TestCmdCallRemove** Java programs, you can get the task again by constructing the **McKey** and creating a new Distributed Task View.

```
McKey tempKey = new McKey(

"com.ibm.mc.client.activity.task.command.McDistCommandDescriptor",
                    "Command Task Name");
```

Be Aware: The test programs referenced above were created for the purpose of testing the Management Central Java Framework, and no attention has been paid to quality GUI programming concepts. You should not use these tests as guides on exactly how to set up your client, but only on how to interact with **Distributed Descriptor** and **View** Objects within the jMC.

For instance, while the jMC provides asynchronous status and results from each endpoint specified in the system group, there is no alternative method for receiving synchronous status or results. After calling the view's execute method, these test programs suspend the main thread until a status update has arrived, after which the main thread is resumed, and execution completes. What this means is that if you use more than one endpoint system, you will lose all status and results information from every system in your system group *except* the one that finishes first.

So, while the test cases will show you how to send and receive data from your central site in the distributed environment of the Management Central Java Framework, it does not give advice as to how to handle that data.

See the section on Plugging Into Operations Navigator for a more robust implementation of handling status and result updates within this asynchronous environment.

Management Central Distributed API Application

Overview

In this chapter you will learn about classes that allow you to call an AS/400 Application Programming Interface(API) on multiple remote AS/400 endpoint systems. You will use classes provided by the AS/400 Java Toolbox in conjunction with classes provided by the Management Central Java Framework. The **McDistApiDescriptor** class with help from the **McDistributedTaskView** and **McDistributedTaskListView** classes provide the Management Central Distributed Task functions allowing a Java program to run a program or service program API on multiple groups of systems and return status and results back to the Central System and the graphical client workstation.

You may choose to use the **ProgramCall**, **ServiceProgramCall**, or **ProgramCallDocument** classes from the AS/400 Java Toolbox to define and construct your API request. Passing one of these objects to the Management Central Distributed API Application, Management Central can send and run the API on the endpoint systems. These classes use the **AS400Message** class to return messages that may have been logged in the job log as a result of the API execution. This message array will be returned in the result for each endpoint system receiving the request.

Interfaces and Flows

Application Designer

As the application designer, you will want to determine whether the **McDistApiDescriptor** class contains the functionality your application needs. Your application would use this function if the interface you are calling on the AS/400 is an Application Programming Interfaces(API) and you require only minimal status and results about the execution of the API.

Some of the specifications of the **McDistApiDescriptor** are:

- Can run a single AS/400 API simulatneously on multiple systems
- Runs asynchronously, meaning a once the task is distributed to the endpoints, each endpoint runs the task in parallel and reports status back upon completion of the task
- Returns a limited defined set of status. It will return Completed when no messages are returned and Failed when any message is returned.
- Returns any output parameters within the ProgramCall, ServiceProgramCall, or ProgramCallDocument resulting object
- Returns AS/400 messages within the ProgramCall, ServiceProgramCall, or ProgramCallDocument resulting object
- API will run under the user profile of the owner of the activity

Gui Developer

The following scenarios describe how to use the **ProgramCall**, **McDistApiDescriptor**, and **McDistributedTaskView** classes. Processing is very similar when using the **ServiceProgramCall** or **ProgramCallDocument** AS/400 Java Toolbox classes.

Classes and Interfaces:

- com.ibm.mc.client.activity.task.api.McApiData
- com.ibm.mc.client.activity.task.api.McDistApiDescriptor
- com.ibm.mc.client.activity.task.api.McEndpApiDescriptor
- com.ibm.mc.server.activity.task.api.McEndpApiAction
- com.ibm.mc.client.activity.task.McDistributedTaskView
- com.ibm.mc.client.activity.task.McDistributedTaskListView
- com.ibm.mc.client.activity.McActivityDescriptorSelectionCriteria
- com.ibm.mc.client.McManageableSelectionCriteria
- com.ibm.as400.access.AS400Message
- com.ibm.as400.access.ProgramCall
- com.ibm.as400.access.ServiceProgramCall
- com.ibm.as400.data.ProgramCallDocument

Scenario 1: Create and Execute a Distributed API Application Task

It is very easy for the GUI developer to use the **McDistApiDescriptor** class to create and execute a distributed API task. Here are the steps to get you started:

1. Create an instance of an AS/400 Java Toolbox **ProgramCall** class and associated parameters.
2. Create an instance of a Distributed API Descriptor specifying the Task Name, Task Owner, Task Description, Sharing, System Group, and the **ProgramCall** object created in step 1. Note that the getSystemGroup method used in the Descriptor's constructor must be supplied by the user to retrieve the list of endpoint systems on which to execute the command.
3. Create an instance of a Distributed Task View specifying the Distributed API Descriptor created in step 2.
4. Tell the Distributed Task View to add the instance of your task so that it can be managed.
5. Call the execute method to distribute and call the API on all the endpoint systems specified in the System Group.

```
    McDistApiDescriptor   distApiDesc = null;
    McDistributedTaskView distApiView = null;

    // Step 1: Create an instance of a ProgramCall object and set associated
parameters
    // Create and/or retrieve AS400 object
    AS400 as400System = getSystem();

    // Create the path to the program.
    QSYSObjectPathName programName = new QSYSObjectPathName("QSYS", "QWCRSSTS",
"PGM");

    // Create the program call object.  Assocate the object with an AS400
object.
    ProgramCall apiSystemStatus = new ProgramCall(as400System);

    // Create the program parameter list.  This program has five
    // parameters that will be added to this list.
    ProgramParameter[] parmlist = new ProgramParameter[5];

    // The AS/400 program returns data in parameter 1.
    parmlist[0] = new ProgramParameter( 64 );

    // Parameter 2 is the buffer size of parm 1.
    AS400Bin4 bin4 = new AS400Bin4( );
    Integer iStatusLength = new Integer( 64 );
    byte[]  statusLength = bin4.toBytes( iStatusLength );
    parmlist[1] = new ProgramParameter( statusLength );

    // Parameter 3 is the status-format parameter.
    byte[] format = McUtilities.stringToByteArray(as400System.getCcsid(),
"SSTS0200");
    parmlist[2] = new ProgramParameter(format);

    // Parameter 4 is the reset-statistics parameter.
    byte[] reset = McUtilities.stringToByteArray(as400System.getCcsid(), "*NO
   ");
    parmlist[3] = new ProgramParameter( reset );

    // Parameter 5 is the error info parameter.
    byte[] errorInfo = new byte[32];
    parmlist[4] = new ProgramParameter( errorInfo, 0 );

    // Set the program to call and the parameter list to the program call
object.
    apiSystemStatus.setProgram( programName.getPath(), parmlist );

    // Step 2: Create an instance of a Distributed API Descriptor
    //         Create the Management Central Distributed API Descriptor task
```

Note: Step 1 above requires you to supply your own AS400 Object via the getSystem method. The only thing it is used for in this example is in converting the Strings into byte array representations. The AS400 Object is necessary so that the conversion routine knows which character set ID (CCSID) to use on the conversion. When executing on the endpoint, the AS400 Object contained within the ProgramCall will be replaced with an object representing the current system. However, the CCSID issue leads to some subtle complexities when dealing with this text conversion.

First, it means that each AS/400 endpoint system in your system group MUST have the same CCSID as the AS400 you use to construct the ProgramParameter list. If it does not, the ProgramParameters may not be interpreted correctly on the endpoint system, causing your program to fail.

Second, it means that a connection must be established to retrieve the CCSID value for some AS400. You can do this either by retrieving the current central system from Operations Navigator, providing an AS/400 system name, user profile, and password with which to connect, or by creating an empty AS400 object, and allowing it to prompt the user for the appropriate information.

Of course, none of these issues arise if you have no need for text-to-byte array conversion as part of your ProgramParameter setup.

Scenario 2: Get a list of Distributed API Application Tasks

If you need to retrieve a list of Distributed API Tasks that would include the task you created in Scenario 1, this next step will show you how. There are only a few steps needed here.

1. Set up the selection criteria to only get tasks of the class **McDistApiDescriptor**
2. Create an instance of the McDistributedTaskListView to manage the list of tasks
3. Ask the Distributed Task List View to return to you a list of Distributed API Tasks

```
  public Vector listTasks()
  {
    McManageableSelectionCriteria selCriteria    = null;
    McDistributedTaskListView     viewList       = null;
    Vector                        retrievedTasks = new Vector();

    // Step 1: Define selection criteria to get a list of Distributed API Tasks
    selCriteria = new McManageableSelectionCriteria(
               "com.ibm.mc.client.activity.task.api.McDistApiDescriptor", // Class
               McManageable.ALL, // Category
               null,             // List of owners (only used if next parameter is
true)
               false,            // Include shared activities
               0);               // Check the last changed date of the activity

    // Step 2: Create a new Task List View to manage your tasks
    viewList= new McDistributedTaskListView(selCriteria);
```

Note: This example will retrieve all the tasks of type **McDistApiDescriptor** and return them in a Vector. The **McDistApiDescriptor** class is the same class used in Scenario 1 step 2 when you created the task.

Scenario 3: Delete a Distributed API Task

If you need to delete a Distributed API task, or a list of them, the Distributed Task View and List View classes provide the methods for you. Deleting a task removes the task from the Management Central

databases and is no longer a manageable task. When working with the Distributed Task View object, you can simply call removeManageable method to delete it.

```
// Step 1: Tell the Distributed Task View to remove the instance of the task
distApiView.removeManageable();

// Or, for a Distributed Task List View
```

Scenario 4: Change a Distributed API Task

To save changes of an existing task on the Central System, the Distributed Task View provides the method for you to use. Prior to calling change, you would have a reference to a task that you previously created or retrieved from a list of tasks. With the reference to the task, you may have the end user modify it by displaying property pages and using the appropriate set methods to update the task instance. When you have the task instance up to date, you can tell the Distributed Task View to store the changes.

```
// Step 1: Tell the Distributed Task View to change the instance of the task
distApiView.changeManageable();
```

Scenario 5: Get asynchronous status and results of a Task

Once you've created your task and called execute, the task will run asynchronously with other activities. If you want to monitor the status of the request, you will want to attach a class to handle when status and results are returned from the endpoint system to you. This class will implement the **McStatusDetailListener** and **McResultDetailListener** interfaces. Note: In the example shown below, the same class implements both these interfaces so we pass in this as the object to handle status and result updates.

```
// Previously you would have retrieved a list of task views and selected the
// one task view that you want to monitor status and results

// Attach to be notified when a Status or Result object is received
view.attachStatusDetailListener(this);
view.attachResultDetailListener(this);

// Display a status window or dialog

// When the user is done with this window or dialog, detach the status and
// result listeners before closing the window
```

```
// Implementing the McStatusDetailListener interface
public void statusUpdate(McStatusEvent event) throws McException
{
  // Get the status object out of the event information
  McStatusIfc status = event.getStatus();

  // If the overall status value indicates the task has finished
  if ( status.getLevel() == McStatusIfc.DistributedAct && status.isFinalized()
)
  {
```

Hints and Tips: The statusUpdate method will be called on a separate thread when status is returned back to the listener. If you want your task to appear to run synchronously, then you can use the suspend method after performing the execute action and place a resume method call here in the statusUpdate method. Be aware, however, that since results are also asynchronous, you have no guarantee that results will be received before status.

```
// Implementing the McResultDetailListener interface
public void resultUpdate(McResultEvent event) throws McException
{
  // Get the result object out of the event information
  McResultIfc result = event.getResult();

  // Since the result object is a hierarchy of results for each Endpoint
System
  // specified in the task, you need to get the results for the specific
Endpoint
  // System to see its details.
  McResultIfc childResult = (McResultIfc)(result.findChild("system1"));

  // Be sure that the result object is an instance of ProgramCall before
  // performing ProgramCall type methods.
  if(childResult != null && childResult.getResultData() instanceof
ProgramCall)
  {
    // Get the ProgramCall object our of the result object
    ProgramCall pgm = (ProgramCall)childResult.getResultData();

    if ( (pgm.getMessageList() != null) && (pgm.getMessageList().length > 0) )
    {
      // Retrieve list of AS/400 messages
```

Terms, Classes, and Interfaces

Thing	Type	What to do with it	Purpose
McDistApiDescriptor	Class	Use	An application will create one of these objects to run an AS/400 Application Programming Interface (API) on multiple systems.
McDistributedTaskView	Class	Use	This class bridges your application to MC Java Framework functions to manipulate tasks and provide additional capabilities that make GUI programming easier. By using the attach and detach capabilities, GUI databeans can be notified directly when a status has changed or when results have been received. Task Actions: - execute - displayScheduleDialog - schedule - cancel View Actions: - addManageable - getManageable - changeManageable - removeManageable Event Listeners: - attachConnectionListener - detachConnectionListener - attachStatusDetailListener - detachStatusDetailListener - attachResultDetailListener - detachResultDetailListener
McDistributedTaskListView	Class	Use	This class bridges your application to MC Java Framework functions to manipulate lists of tasks and provide additional capabilities that make GUI programming easier. By using the attach and detach capabilities, GUI classes can be notified directly when a task has been created, changed, updated, and deleted. Distributed Task Manager: - getManageableViews - removeManageableList Event Listeners: - attachManageableListener - detachManageableListener
McManageableSelectionCriteria	Class	Use	Use this class in conjunction with the

			McDistributedTaskListView to specify the type of tasks to retrieve. The selection criteria allows you to specify Type, Category, Sharing, etc.
McActivityDescriptorSelectionCriteria	Class	Use	Like McManageableSelectionCriteria, you use this class in conjunction with the McDistributedTaskListView to specify the type of tasks to retrieve. This class extends the base to include selection criteria to subset activities based on their status values.
ProgramCall	Class	Use	Provided by the AS/400 Java Toolbox: Contains the AS/400 program API and parameters to call on all the remote systems. Output parameters will be returned in a resulting ProgramCall object. This object will also be used to return AS/400 messages if the execution of the API resulted in any messages.
ServiceProgramCall	Class	Use	Provided by the AS/400 Java Toolbox: Contains the AS/400 service program API and parameters to call on all the remote systems. Output parameters will be returned in a resulting ServiceProgramCall object. This object will also be used to return AS/400 messages if the execution of the API resulted in any messages.
ProgramCallDocument	Class	Use	Provided by the AS/400 Java Toolbox: Contains the AS/400 program API or service program API and parameters to call on all the remote systems. Output parameters will be returned in a resulting ProgramCallDocument object. This object will also be used to return AS/400 messages if the execution of the API resulted in any messages.

Programming Examples

In CMVC there are a number of test programs available at:

as400a\v5r1m0t.ss03\int\cmvc\java.pgm\yps.ss03\com\ibm\app\client

- TestApiPgm
- TestApiPgmCallCreate
- TestApiPgmCallAttach
- TestApiPgmCallExecute
- TestApiPgmCallRemove

The **TestApiPgm** Java program is an all inclusive test program that will create a new task, attach for status and result notifications, execute the task, process status and result events, and remove the task when completed.

The rest of the test programs break the entire test into controllable pieces. The **TestApiPgmCallCreate** Java program will create a new task. The **TestApiPgmCallAttach** Java program will associate itself to the task so when it executes it can receive status and result notifications. The **TestApiPgmCallExecute** Java program will kick off the execution of the task and will also receive status and result notifications. The **TestApiPgmCallRemove** Java program will delete the task from the Management Central Task data base on the AS/400.

In these examples it is important to understand the **McKey** concept. When a new task is created, a key is created to uniquely identify the task This key is made up of three parts: the task class, the task name, and the user who owns the task. In the **TestApiPgmCallCreate** Java program the key is created and assigned when you create a new instance of your task. This happens when you instantiate a new **McDistApiDescriptor** and perform an addManageable.

```
distApiDesc = new McDistApiDescriptor("McDistApiTask_Name",       // Name
                                      "McDistApiTask_Descriptor", // Description
                                      McManageable.NONE,          // Sharing
                                      getSystemGroup(),           // System Group
                                      apiSystemStatus,            // ProgramCall
                                      null);                      // Category

distApiView = new McDistributedTaskView(distApiDesc);

distApiView.addManageable();
```

Notice that when you construct a new **McDistApiDescriptor**, you specified two out of the three essential parts of the key:

1. Task Class = McDistApiDescriptor
2. Task Name = “DistApiDesc-TestTask”

The owner is determined by the Management Central Java Framework.

Now in the **TestApiPgmCallAttach**, **TestApiPgmCallExecute**, and **TestApiPgmCallRemove** Java programs, you can get the task again by constructing the **McKey** and creating a new Distributed Task View.

```
      McKey tempKey = new
McKey("com.ibm.mc.client.activity.task.api.McDistApiDescriptor",
                                "DistApiDesc-TestTask");
      distApiView= new McDistributedTaskView(tempKey);
```

Be Aware: The test programs referenced above were created for the purpose of testing the Management Central Java Framework, and no attention has been paid to quality GUI programming concepts. You should not use these tests as guides on exactly how to set up your client, but only on how to interact with **Distributed Descriptor** and **View** Objects within the jMC.

For instance, while the jMC provides asynchronous status and results from each endpoint specified in the system group, there is no alternative method for receiving synchronous status or results. After calling the view's execute method, these test programs suspend the main thread until a status update has arrived, after which the main thread is resumed, and execution completes. What this means is that if you use more than one endpoint system, you will lose all status and results information from every system in your system group *except* the one that finishes first.

So, while the test cases will show you how to send and receive data from your central site in the distributed environment of the Management Central Java Framework, it does not give advice as to how to handle that data.

See the section on Plugging Into Operations Navigator for a more robust implementation of handling status and result updates within this asynchronous environment.

Advanced Features

The following scenarios are provided to supplement the information in each of the preceding application sections. They are considered advanced features only because they aren't necessary to use the Command Call or API applications provided by the Management Central Java Framework. However, they are quite useful, and are provided here as a reference.

All examples are shown using the Command Call Application as the base, but since these techniques apply to all Tasks, they are valid features of the Api application as well.

Saving Endpoint Systems and System Groups

The Endpoint system and System Group concepts are key elements of any Distributed Application within the Management Central Java Framework. The jMC also provides the tools for you to store this data persistently on the central site to be used or reused at a later time. Once stored on the Central System, it can be optionally viewed, changed, removed, and shared by all the operators and administrators connecting to that Central System.

This persistently stored data is called a Definition in the jMC. Definitions are discussed in depth in the full version of this How To document, but for the purposes of this discussion, we'll only cover how to persistently store Endpoint Systems and System Groups.

Classes and Interfaces:

- com.ibm.mc.client.definition.McDefinition
- com.ibm.mc.client.definition.McDefinitionView
- com.ibm.mc.client.definition.McEndpointSystem
- com.ibm.mc.client.definition.McSystemGroup
- com.ibm.mc.client.definition.McEndpointSystemSelectionCriteria
- com.ibm.mc.client.definition.McSystemGroupSelectionCriteria

Scenario 1: Creating Definition Instances

In Scenario 1, you create and manage an instance of your Definition. This instance can either be a **McEndpointSystem** or a **McSystemGroup** (consult the JavaDoc for class constructor specifications). Basically there are three steps in creating a new definition:

1. Create an instance of your definition. This could be triggered from the user interface where the user specifies different properties.
2. Create an instance of a Definition View to manage the instance of your definition
3. Tell the Definition View to persistently store the instance of your definition to the definition database on the Central System using the addManageable method.

```
public void newDefinition()
{
   McEndpointSystem endpSys  = null;
   McDefinitionView defnView = null;

   // Step 1: Create an instance of the MyCommandDefinition class
   endpSys = new McEndpointSystem(
                    "SystemName",             // System Name
                    "System description",     // Description
                    McManageable.READ,        // Sharing
                    "9.5.179.19");               // IP address

   // Step 2: Create a Definition View object
   defnView = new McDefinitionView(endpSys);
```

Scenario 2: Retrieve your Definitions

If you need to retrieve a Definitions that you created in Scenario 1, this next scenario will give you the basics. There are only a few steps needed here:

1. Set up the selection criteria to only get the definition that matches the name of your application's definition, **McEndpointSystem** in this case. Notice that instead of using McManageableSelectionCriteria to select your endpoint, you'll use **McEndpointSystemSelectionCriteria**. A similar class, called **McSystemGroupSelectionCriteria**, exists for selecting system groups.
2. Create an instance of a Definition View to manage your definition.
3. Ask the Definition View to return you an endpoint system that matches the selection criteria you specified in step 1.

```
public McEndpointSystem getSystem()
{
  McEndpointSystemSelectionCriteria selCriteria = null;
  McDefinitionView                  defnView    = null;

  // Step 1: Define selection criteria for a list of your definitions
  selCriteria = new McEndpointSystemSelectionCriteria("SystemName");

  // Step 2: Create a new Definition List View object to manage your definitions
  defnView = new McDefinitionView(selCriteria);

  // Step 3: Ask the Definition List View to return objects that match your criteria
  McEndpointSystem endpSys = (McEndpointSystem)defnView.getManageable();
```

Scenario 3: Deleting Definition Instances

If you need to delete a definition, or a list of definitions, the same View and List View wrappers provide the methods to use. The removeManageable for a single definition view and removeManageables for a

list of definition views are methods available on the **McDefinitionView** and **McDefinitionListView** classes.

```
// Step 1: Tell the Definition View to remove your Definition
//         It will be deleted from the persistent store on the
Central
//         System and will no longer be available.
```

Scenario 4: Changing an existing Definition Instance

To change an existing definition, the View again provides the method for you to use. Prior to calling change, you would have a reference to a definition that you previously created or retrieved from a list. With the reference to the definition, you may have the end user modify it by displaying a set of property pages and using the appropriate set methods to update instance of the definition. When you have the definition instance updated, you can tell the View to store the changes.

```
// Step 1: Call the set methods to update the Definition
endpSys.setAddress("255.255.255.255");

// Step 2: Tell the Definition View to change your Definition with the updates
made
```

Get asynchronous status updates for List of Tasks

By implementing the **McManageableListener** interface, you can be notified when a new task has been created, changed, updated, or deleted. This is most useful when maintaining a list of tasks, and you wish to be notified whenever tasks are added, removed, updated, or changed. This list of tasks is specified using selection criteria so that you are not notified when just any task is created, but only those tasks that meet your selection subset. This code is identical to the code used to retrieve a list of Tasks based on a selection subset, but we add a fourth step here to attach the current class as the ManageableListener. This interface, along with the implemented update, change, and remove methods, allows the Management Central Java Framework to send you a notification when a Task has been updated.

```
public class MyTaskList implements McManageableListener
{
  public void getList() {
    McManageableSelectionCriteria selCriteria    = null;
    McDistributedTaskListView     viewList       = null;
    Vector                        retrievedTasks = null;

    // Step 1: Define selection criteria to get a list of Distributed Command
Tasks
    selCriteria = new McManageableSelectionCriteria(

"com.ibm.mc.client.activity.task.command.McDistCommandDescriptor",
                  "McManageable.ALL",    // category
                  null,            // Owner List (only used when next parm is true)
                 false,            // use sharing
                  0);                    // last changed date

    // Step 2: Create a new Task List View to manage your tasks
    viewList = new McDistributedTaskListView(selCriteria);

    // Step 3: Ask the Distributed Task Manager for a list of Distributed Command
Tasks
    retrievedTasks = viewList.getManageableViews();

    // Step 4: Attach this class, which implements the McManageableListener
interface,
    //         to handle any notifications.
    viewList.attachManageableListener(this);
  }

  // The following methods are required as an implementation of
McManageableListener

  public void manageableAdded(McManageableEvent event) throws McException {
    // Insert code to handle when a new task has been created
  }
```

Private Descriptors

By default, all Descriptors are public. Mainly, this means that once the manageable is managed by the jMC, it will be stored persistently in a database on the server. Any user that has appropriate authority can retrieve that descriptor off the server by setting up a SelectionCriteria object that meets some criteria of the descriptor, creating a list view with the selection criteria, and calling `getManageableViews` on the listView.

Private descriptors, on the other hand, are never stored persistently, and may not be retrieved off the server once it's created, even by the owner. Even if a user specifies a selection criteria that exactly matches the private descriptor, it will not be returned in their list.

In the following example, the descriptor is created as normal, but Step 1a is added to make the descriptor private. Without this step, anyone with the appropriate authority could retrieve the task from the server, and possible update it.

```
McDistCommandDescriptor distCommandDesc = null;
McDistributedTaskView distCommandView   = null;

// Step 1: Create an instance of a CommandCall object and set the command
CommandCall cmdToRun = new CommandCall();
cmdToRun.setCommand("CRTLIB USRLIB");

// Step 2: Create an instance of a Distributed Command Descriptor
distCommandDesc = new
         McDistCommandDescriptor("Command Task Name",            // Name
                                 "Command Task Description",   // Description
                                 McManageable.NONE,            // Category
                                 getSystemGroup(),             // System Group
                                 cmdToRun,                     // CommandCall
Object
                                 null);                 // Category

// Step 2a: Make descriptor private
distCommandDesc.setPrivate(true);
```

Public Descriptor Sharing

In this scenario, we added Step 2a to change the Descriptor's sharing value. Sharing lets the owner specify whether other users can view or change the contents of the descriptor. Only public descriptors can be shared.

```
McDistCommandDescriptor distCommandDesc = null;
McDistributedTaskView distCommandView   = null;

// Step 1: Create an instance of a CommandCall object and set the command
CommandCall cmdToRun = new CommandCall();
cmdToRun.setCommand("CRTLIB USRLIB");

// Step 2: Create an instance of a Distributed Command Descriptor
distCommandDesc = new
         McDistCommandDescriptor("Command Task Name",            // Name
                                 "Command Task Description",   // Description
                                 McManageable.NONE,            // Sharing
                                 getSystemGroup(),             // System Group
                                 cmdToRun,                     // CommandCall
Object
                                 null);                 // Category

// Step 2a: Set Sharing to Full
distCommandDesc.setSharing(McManageable.FULL);
```

By default, there is no sharing of descriptors, but by setting the sharing value of this McDistCommandDescriptor to McManageable.FULL, all users will be able to retrieve the descriptor

using the listView’s getManageableViews method, and will also be able to make and store changes to the server using the changeManageable method on the View.

Valid sharing values and their meanings are:

McManageable.NONE	This is the default sharing value. No users will be able to view the descriptor.
McManageable.READ	All users will be able to view but not change, the descriptor.
McManageable.FULL	All users will be able to view and change the descriptor.

Auto Increment

In this scenario, we added Step 2a to set auto increment to true. Auto increment allows you to create multiple instances of a task without worrying about a name conflict. The first time this is run, it will create a task with a name of "Command Task Name". By adding step 2a, the second time this is run the Management Central Java Framework will automatically identify that the name "Command Task Name" already exists and increment the name to "Command Task Name(2)". The third time it will be "Command Task Name(3)" and so on.

```
McDistCommandDescriptor distCommandDesc = null;
McDistributedTaskView distCommandView   = null;

// Step 1: Create an instance of a CommandCall object and set the command
CommandCall cmdToRun = new CommandCall();
cmdToRun.setCommand("CRTLIB USRLIB");

// Step 2: Create an instance of a Distributed Command Descriptor
distCommandDesc = new
         McDistCommandDescriptor("Command Task Name",            // Name
                                 "Command Task Description",    // Description
                                 McManageable.NONE,             // Category
                                 getSystemGroup(),              // System Group
                                 cmdToRun,                      // CommandCall
Object
                                 null);                 // Category

// Step 2a: Set Auto Increment On
distCommandDesc.setAutoIncrement(true);
```

Categories

When the same task class needs to be used for multiple purposes, categories are used to distinguish between them. In this command task example, you may have the need for both Backup-type tasks and Restore-type tasks. When you create an instance of your task, you can specify a Category to use like "MyTask-Backup" instead of creating a separate task class for each. This is done in Step 2b in the following example.

```
McDistCommandDescriptor distCommandDesc = null;
McDistributedTaskView   distCommandView = null;

// Step 1: Create an instance of a CommandCall object and set the command
CommandCall cmdToRun = new CommandCall();
cmdToRun.setCommand("CRTLIB USRLIB");

// Step 2: Create an instance of a Distributed Command Descriptor
distCommandDesc = new
         McDistCommandDescriptor("Command Task Name",           // Name
                                 "Command Task Description",    // Description
                                 McManageable.NONE,             // Category
                                 getSystemGroup(),              // System Group
                                 cmdToRun                       // CommandCall
Object
                                 null);                  // Category

// Step 2b: Set the Task Category
//          Optionally, specify this value on the descriptor constructor above
distCommandDesc.setCategory("MyTask-Backup");
```

If you specify a category when you create an instance of your Distributed Command Call task, you can specify that same category in your selection criteria to only get tasks that match. This allows you to retrieve a list of only Backup-type tasks or Restore-type tasks even though they both share the same **McDistCommandDescriptor** class.

```
McManageableSelectionCriteria selCriteria    = null;
McDistributedTaskListView     viewList       = null;
Vector                        retrievedTasks = new Vector();

// Step 1: Define selection criteria to get a list of Distributed Command
Tasks
selCriteria = new McManageableSelectionCriteria(

"com.ibm.mc.client.activity.task.command.McDistCommandDescriptor",
                  "MyTask-Backup",// Category
                  null,           // Owner List (only valid if next parm is true)
                  false,          // use sharing
                  0);                    // last changed date
```

Schedule Your Task

It is very easy for the GUI developer to use the **McDistCommandDescriptor** class to create and schedule a distributed command call task. To do this we added steps 5 and 6 to what has previously been covered.

1. Create an instance of an AS/400 Java Toolbox CommandCall object and set the command.
2. Create an instance of a Distributed Command Descriptor specifying the Task Name, Task Owner, Task Description, Sharing, and a System Group; then set the command using the CommandCall object created in step 1.
3. Create an instance of a Distributed Task View object specifying the Distributed Command Descriptor created in step 2.
4. Tell the Distributed Task View to add the instance of your task so that it can be managed.
5. Construct a **McScheduleInfo** object using a description of your activity, and "execute" as the scheduled method, and set the schedule information by prompting the user with the Management Central Schedule Dialog or a supported Business Partner Scheduler.
6. Call the schedule method to schedule the task on the Central System.

Note: This scenario is very similar to distributing a command call to run on multiple endpoints. The difference is instead of calling execute, you now need to gather scheduling information as the fifth step and call schedule as an additional step.

```
    McDistCommandDescriptor distCommandDesc = null;
    McDistributedTaskView   distCommandView = null;

    // Step 1: Create an instance of a CommandCall object and set the command
    CommandCall cmdToRun = new CommandCall();
    cmdToRun.setCommand("CRTLIB USRLIB");

    // Step 2: Create an instance of a Distributed Command Descriptor
    distCommandDesc = new
             McDistCommandDescriptor("Command Task Name",           // Name
                                     "Command Task Description",   // Description
                                     McManageable.NONE,            // Category
                                     getSystemGroup(),             // SystemGroup
                                     cmdToRun,                     // CommandCall
Object
                                     null);                 // Category

    // Step 3: Create an instance of a Distributed Task View object
    //         specifying the Distributed Command Descriptor.
    distCommandView = new McDistributedTaskView(distCommandDesc);

    // Step 4: Tell the Management Central Java Framework Task Manager to add this
    //         Distributed Command Descriptor to manage.
    distCommandView.addManageable();
```

Get list of Scheduled Distributed Tasks

If you need to retrieve a list of previously Scheduled Distributed Tasks, this next step will show you how. Refer to the previous section where you Scheduled your task to execute at a later date. If that

task is currently managed by the Management Central Java Infrastructure, it will be returned in your list of scheduled tasks below. There are only a few steps needed here.

1. Set up the selection criteria to only get scheduled tasks of the class **McDistCommandDescriptor**
2. Create an instance of the **McDistributedTaskListView** to manage the list of tasks
3. Ask the Distributed Task Manager to return to you a list of Distributed Command Call Tasks

```
    McActivityDescriptorSelectionCriteria selCriteria = null;
    Vector retrievedSchedTasks = new Vector();
    int[] statusList = {McStatusIfc.Scheduled};

    // Step 1: Define selection criteria to get a list of  Scheduled Distributed
    //         Command Call Tasks
    selCriteria = new McActivityDescriptorSelectionCriteria(

"com.ibm.mc.client.activity.task.command.McDistCommandDescriptor",
                      McManageable.ALL,   // Category
                      null,               // OwnerList (only valid if next parm is
true)
                      false,                     // useSharing
                      0,                  // last changed date
                      statusList);        // StatusList - ours contains only
```

Note: This time you needed to use the **McActivityDescriptorSelectionCriteria** class instead of the **McManageableSelectionCriteria**. The activity selection criteria extends the capabilities of the manageable selection criteria to include status information. This allows you to indicate that you only want to receive activities that are in a particular status, such as active, completed, or in this case scheduled.

Handling Exceptions

In this scenario, a **com.ibm.mc.client.util.McException** is caught and interrogated to determine the cause of an error. If, for whatever reason, an Exception is thrown during processing, the Management Central Java Framework will always attempt to catch the Exception, whether it was thrown initially by some jMC method or by any other Java method, and package it into a McException. This McException may then be caught and re-thrown with additional information from the caller of the errant message, and so on until the Exception is finally re-thrown remotely to the client. Therefore, when this McException is returned to the client, it may have multiple nested Exception objects within it.

In your catch block, you may interrogate the McException with the containsErrorID method of McException to determine if a particular error ID. This identifier must be either an ID defined in the **McService** class, or a class name of a predefined Java Exception class. (McService refers to the logging of service messages, or job logging on the AS/400, and is not to be confused with the services we've defined as activities in the jMC). Alternatively, the error can be output to the client for informational purposes with the printStackTrace method. Consult the JavaDoc on McException for

further information on how to use the McException class, and McService to view predefined error ID strings.

```
try
{
    view.addManageable();
}
catch( McException e )
{
    if( e.containsErrorId("java.sql.MCJS_MGBL_DUPKEY") )
       return "Key Error";
    else if( e.containsErrorId("java.io.IOException") )
       return "I/O Error";
    else
    {
```

Tracing Messages

The Management Central Java Framework provides a default tracing mechanism to make it easier for you to trace messages. Using class **McTrace** in package com.ibm.mc.client.util, tracing messages to a file becomes a one-step process. For instance, when retrieving instances of your Distributed Command Tasks off the server (as you did in the Distributed Command Call Application Section of this document, Scenario 2), you may want to trace certain elements of the execution. Only a few steps are needed here:

1. Initialize trace with the file name you wish to trace to, and the level of data you would like to trace. Valid values for level are Error, Warning, Information, and Diagnostic. If trace level is stet to Error (the default), only messages with Error severity will be logged; if trace level is Information, all Informational, Warning, and Error messages will be logged.
2. Execute your Management Central function.
3. Check trace level, and trace appropriate messages

```
    McManageableSelectionCriteria selCriteria    = null;
    Vector                        retrievedTasks = new Vector();

    // Step 1: Initialize trace
    String fName = "C:\\MGTC.Java.Service.Log";
    McTrace.setFileName(McManageable.MCCOMPONENTNAME, fName);
    McTrace.setTraceLevelOn(McTraceable.INFORMATION);

    // Step 2: Execute Management Central Function
    // Define selection criteria to get a list of Distributed Command Tasks
    selCriteria = new McManageableSelectionCriteria(
                "com.ibm.mc.client.activity.task.command.McDistCommandDescriptor", //
Class
                McManageable.ALL,  // Category
                null,              // List of owners (only used if next parameter is
true)
                false,             // Include shared activities
                0);                // Last changed date of the activity

    // Create a new Task List View to manage your tasks
    McDistributedTaskListView viewList = new McDistributedTaskListView(selCriteria);

    // Step 3: Tracing an informational message
    if( McTrace.isTraceInformationOn() )
        McTrace.logInformation( getClass().getName(), "Executing getList from
server" );

    try {
        retrievedTasks = viewList.getManageableViews();

    } catch( McException mce ) {
       // Step 3: Tracing an Error message
```

This will trace the number of Distributed command Call Tasks that were found on the server, and that match your selection criteria, or alternatively, if an exception occurs, then the exception will be traced.

Additional Utilities

Many convenience classes and methods exist to allow you to do common tasks within the Management Central Java Framework. All classes discussed here reside in the com.ibm.mc.client.util package. If you find you're writing your own convenience methods for tasks you need to execute in multiple places within your code, consult the JavaDoc for these classes - chances are you'll find exactly what you're looking for.

McUtilities	Contains data management utilities. Some make byte array manipulation easier for the user; some simplify serialization and deserialization; some are for data conversion.
McMethodThread	Useful for applications that that wish to process method requests on a privately maintained thread but wish to abstract the details of thread management. Class can be used to queue, de-queue and invoke methods.
McMethodQueue	Provides a default method queuing mechanism. Used alone, it only provides standard queue functionality, but when used in conjunction with a McMethodThread, queued methods can be automatically invoked by the jMC.