Getting started with Microsoft Robotics Studio

By Claus Andersen (claus.andersen@teknologisk.dk)

Contents

Preface

This document will guide you through the steps required to get started with Microsoft Robotics Studio (MSRS). The guide covers how to acquire and install MSRS and how to start a new project. First the basics of a MSRS project will be explained, and next the fundamental service communication is demonstrated through example projects.

The MSRS version 1.5 was released at the time of creating this document and has therefore been used throughout the document.

Prerequisites

It is assumed that Microsoft Visual Studio is installed before proceeding with any of the steps described in the document. A full version of Visual Studio is not required and the express edition will therefore be sufficient. Microsoft Visual C# 2005 Express Edition can be downloaded from Microsoft's Developers

Network at http://msdn2.microsoft.com/en-us/express/aa700756.aspx or via www.msdn.com and navigating to the Visual Studio Express download site.

Before reading this guide you should be familiar with C# and have a basic knowledge of web programming, HTTP, XML and XSLT.

Vocabulary

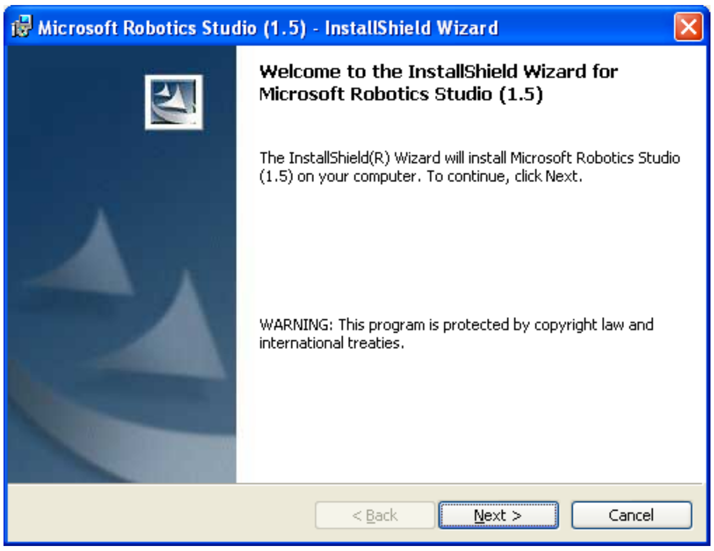
CCR	Concurrency and Coordination Runtime. The CCR is at the base of all MSRS services and provides concurrency programming facilities free from many of the usual hassles of multitasking. It is suitable but not specifically made for robotics. CCR requires the .Net 2.0 Framework.
DSS	Decentralized System Services. DSS is used for communication between services in MSRS systems. The DSS allows the developer to create distributed services. Communication between DSS services is location transparent.
DSSP	DSS Protocol is used in DSS for communication between services. DSSP uses both SOAP[1] and REST[2].
(CCR) Port	A messages queue. The most fundamental communication channel in CCR and DSS. It is used for sending messages both within a service and between services.
Service handler	A method that can be executed on the CCR. The scheduling of a handler is often triggered by a message arriving at a port.
APM	Asynchronous Programming Model. In .Net it is the method for initializing long-term actions on a separate thread and allowing them to report back when their work is completed, thereby freeing the main thread in the meantime.

Obtaining and Installing Microsoft Robotics Studio

All relevant MSRS information and downloads can be found at http://msdn.microsoft.com/robotics/. Download the test version of MSRS (current version is 1.5 Refresh) and run the installation file. At the MSRS website you can also find several updates and add-ons as well as tutorials and other developer resources. Although they are not discussed in this guide, many of them are quite relevant for understanding and utilizing MSRS to its full potential.

Once the installation starts, you should be able see the following window:

[1] Simple Object Access Protocol, see http://www.w3.org/TR/soap for the latest version of the SOAP specification
[2] Representational State Transfer, Roy Fielding, http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm , Ch.5

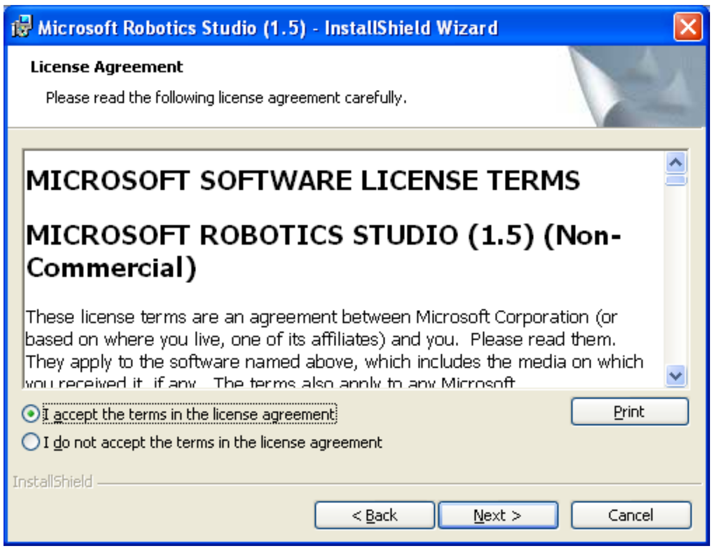


Welcome to the InstallShield Wizard for Microsoft Robotics Studio (1.5)

The InstallShield(R) Wizard will install Microsoft Robotics Studio (1.5) on your computer. To continue, click Next.

WARNING: This program is protected by copyright law and international treaties.

Click “Next” to proceed.



License Agreement

Please read the following license agreement carefully.

MICROSOFT SOFTWARE LICENSE TERMS

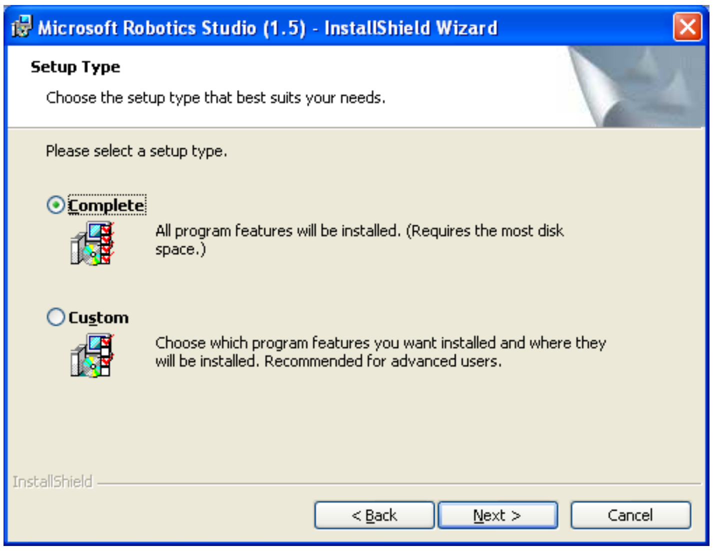
MICROSOFT ROBOTICS STUDIO (1.5) (Non-Commercial)

These license terms are an agreement between Microsoft Corporation (or based on where you live, one of its affiliates) and you. Please read them. They apply to the software named above, which includes the media on which

[✓ I accept the terms in the license agreement]
I do not accept the terms in the license agreement

Accept the license agreement by selecting “I accept the terms in the license agreement” and press the Next button.

The next dialog allows you to customize the installation.



Setup Type

Choose the setup type that best suits your needs.

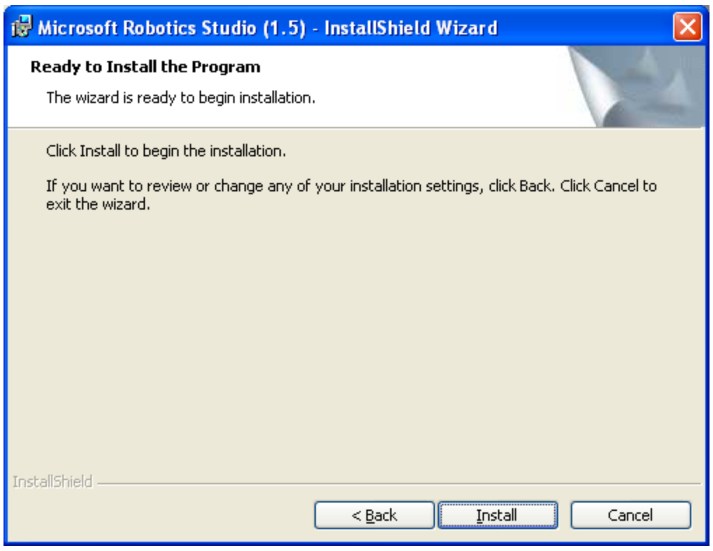
Please select a setup type.

[✓ Complete] All program features will be installed. (Requires the most disk space.)

Custom: Choose which program features you want installed and where they will be installed. Recommended for advanced users.

Custom (Customization) is not recommended. Select “Complete” and press “Next”.

The next dialog allows you to review previous selections.



Ready to Install the Program

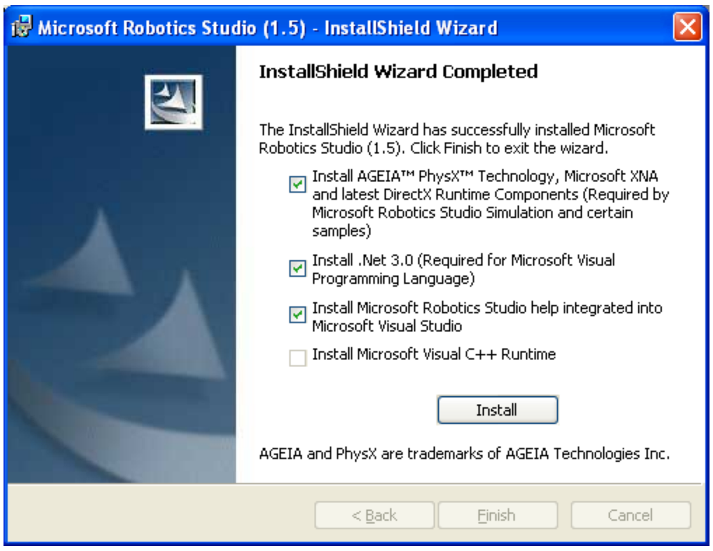
The wizard is ready to begin installation.

Click Install to begin the installation.

If you want to review or change any of your installation settings, click Back. Click Cancel to exit the wizard.

Simply press “Install”.

Once the core modules of Robotics Studio have been installed, the following dialog will appear:



Microsoft Robotics Studio (1.5) - InstallShield Wizard

InstallShield Wizard Completed

The InstallShield Wizard has successfully installed Microsoft Robotics Studio (1.5). Click Finish to exit the wizard.

- [✓] Install AGEIA™ PhysX™ Technology, Microsoft XNA and latest DirectX Runtime Components (Required by Microsoft Robotics Studio Simulation and certain samples)
- [✓] Install .Net 3.0 (Required for Microsoft Visual Programming Language)
- [✓] Install Microsoft Robotics Studio help integrated into Microsoft Visual Studio
- [ ] Install Microsoft Visual C++ Runtime

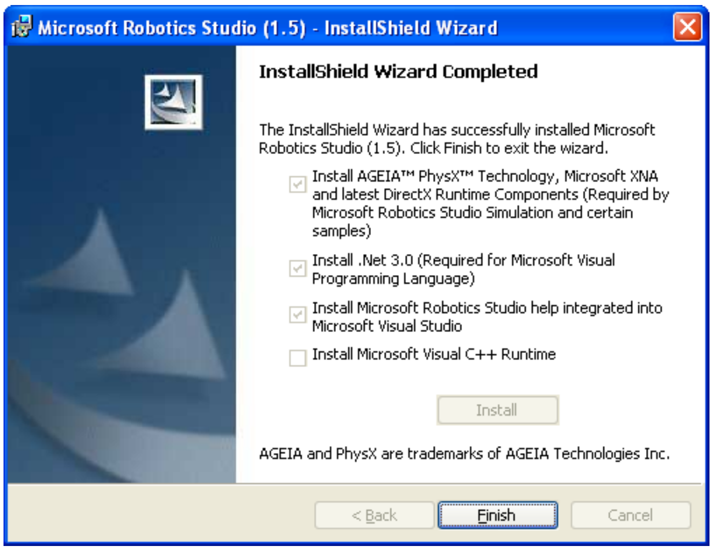
Install

AGEIA and PhysX are trademarks of AGEIA Technologies Inc.

< Back | Finish | Cancel

If the .Net 3.0 framework is not already installed on your computer, please place a tick in the upper three tick boxes. Or just tick the “Install AGEIA…” and “Install Microsoft Robotics…” boxes.
Press “Install” to begin installation of the selected components.

When the selected components have been installed the following dialog appears:



Microsoft Robotics Studio (1.5) - InstallShield Wizard

InstallShield Wizard Completed

The InstallShield Wizard has successfully installed Microsoft Robotics Studio (1.5). Click Finish to exit the wizard.

- [✓] Install AGEIA™ PhysX™ Technology, Microsoft XNA and latest DirectX Runtime Components (Required by Microsoft Robotics Studio Simulation and certain samples)
- [✓] Install .Net 3.0 (Required for Microsoft Visual Programming Language)
- [✓] Install Microsoft Robotics Studio help integrated into Microsoft Visual Studio
- [ ] Install Microsoft Visual C++ Runtime

Install

AGEIA and PhysX are trademarks of AGEIA Technologies Inc.

< Back | Finish | Cancel

Press “Finish” to complete the installation.

The installation of Microsoft Robotics Studio and its components is now completed.

Creating a Robotics Project

There are two ways of creating a new project: use either a command line tool or the Microsoft Visual Studio project wizard. The outputs generated from the two options are identical.
Both C# and Visual Basic projects can be created. This guide only demonstrates C#.

Note that the service name must not end with the word “service” as in for instance “MyService”. If it does, “MyService” would be cropped to “My” in some but not all places!

Using the command prompt

MSRS contains a command line tool called *DssNewService.exe* which can be accessed through the MSRS command prompt. The prompt can be started via the Microsoft Robotics Studio menu in the Windows Start menu.
In the command prompt, change to the directory where the new project should be created.
DssNewService accepts a number of switches but only one is required to create a C# project, and that is /service used to provide the service name.

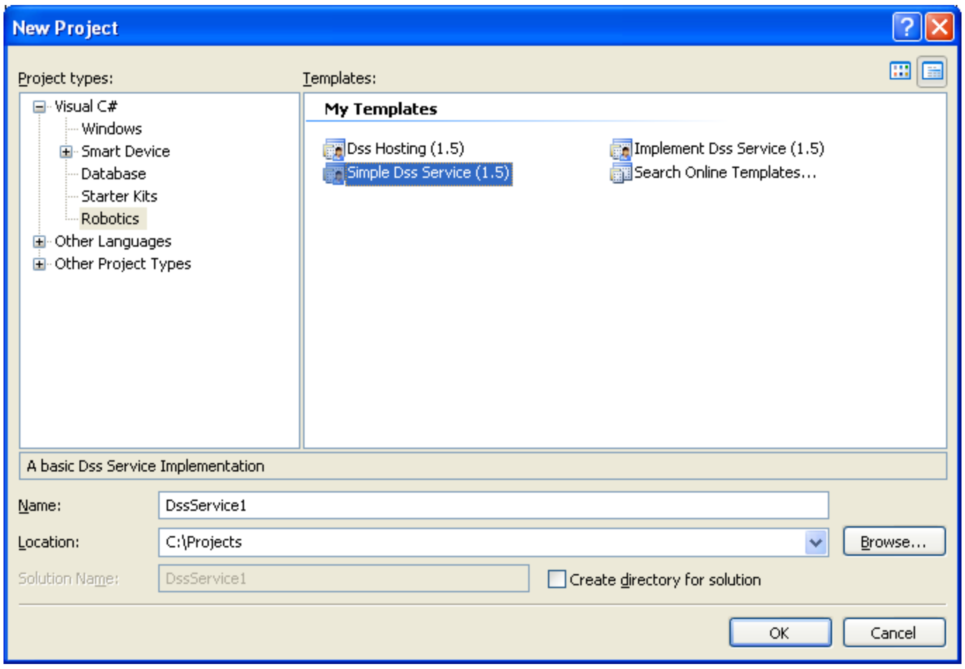
To create a new service called MyProject, type:

```
DssNewService /service:MyProject
```

A new folder called MyProject will be created at the current location and will contain the necessary files, including the Visual Studio solution file.

Using Visual Studio

If Visual Studio has been installed prior to installing MSRS, the Visual Studio’s new project wizard should now have an option for creating robotics projects.


New Project

Project types:
- Visual C#
  - Windows
  - Smart Device
  - Database
  - Starter Kits
  - Robotics
- Other Languages
- Other Project Types

Templates:

My Templates
- Dss Hosting (1.5)
- Simple Dss Service (1.5)
- Implement Dss Service (1.5)
- Search Online Templates...

A basic Dss Service Implementation

Name: DssService1

Location: C:\Projects

Solution Name: DssService1

Create directory for solution

By selecting this project type, several project templates appear. Select the “Simple Dss Service (1.5)” template, specify a project name and location and press the OK button. This creates and opens a new project.

Building the Project

The project you just created is now ready to be built.
As described previously, the task can be performed by using a command line tool or Visual Studio.

Using the command prompt

Open a Visual Studio command prompt and switch to the project directory.
Then type

```
MsBuild MyProject.sln /p:configuration=debug
```

Using Visual Studio

Once the project is opened, simply press the F6 key.

Hosting the Service

Once the project has been built, the service is ready to be hosted. This activity is performed by the *DssHost.exe* tool which is activated by using the command prompt or Visual Studio.

Using the command prompt

Use a MSRS command prompt to change to the project directory and type

```
DssHost /m:MyProject.manifest.xml /p:50000
```

The switch /p:50000 specifies that DssHost must use port 50000, and it is also typically the one used. It is the TCP port used to access the host and services which it is running.

Using Visual Studio

Once the project is opened, simply press the F5 key (Debug) and the project will be built. Afterwards it is hosted by the DssHost which is automatically started by Visual Studio.
Port 50000 is selected by default for the DssHost.

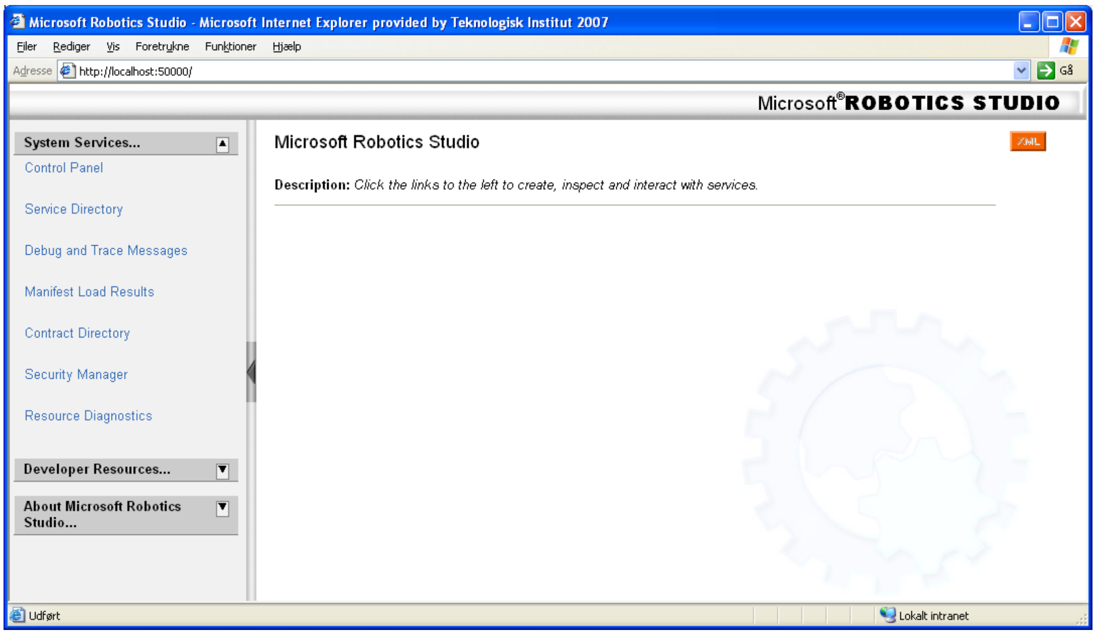
Both when DssHost is started from the command prompt or through Visual Studio, it can be stopped by pressing Ctrl + C. This shuts down any running services and the host.

When hosted by DssHost, the service created can be reached by navigating to http://localhost:50000/MyProject in a web browser.

The created project only provides the basic DSSP Get request which returns a SOAP enveloped XML document containing the fundamental state information.

Built-in Services

When the DssHost is started, some built-in services are automatically loaded to aid debugging your service. Just like any other DSS service these services can be reached through a web browser. Navigate to http://localhost:50000 which will provide the main page for debugging services.



Microsoft Robotics Studio - Microsoft Internet Explorer provided by Teknologisk Institut 2007

Filer Rediger Vis Foretrukne Funktioner Hjælp

Adresse http://localhost:50000/

Microsoft® ROBOTICS STUDIO

System Services...
- Control Panel
- Service Directory
- Debug and Trace Messages
- Manifest Load Results
- Contract Directory
- Security Manager
- Resource Diagnostics

Developer Resources...

About Microsoft Robotics Studio...

Microsoft Robotics Studio

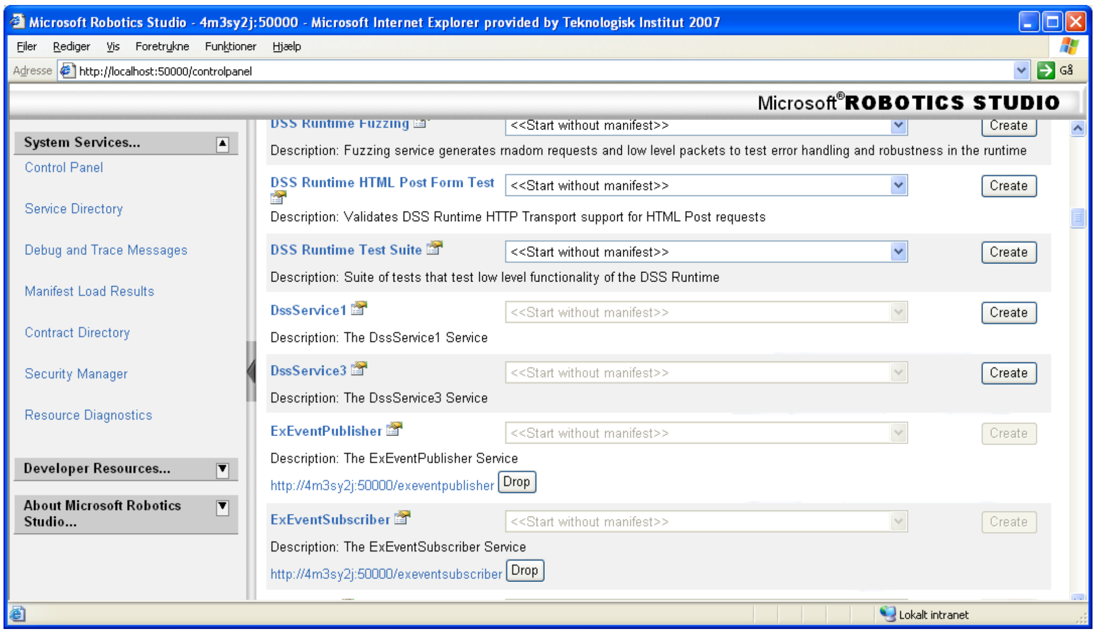
Description: *Click the links to the left to create, inspect and interact with services.*

Udført — Lokalt intranet

This is where the debugging information can easily be reached.

Three of these services are shown below. They provide basic information on the services running on the host right now and the services in the contract directory.

The control panel



Microsoft Robotics Studio - 4m3sy2j:50000 - Microsoft Internet Explorer provided by Teknologisk Institut 2007

Filer Rediger Vis Foretrukne Funktioner Hjælp

Adresse http://localhost:50000/controlpanel

Microsoft® ROBOTICS STUDIO

System Services...
- Control Panel
- Service Directory
- Debug and Trace Messages
- Manifest Load Results
- Contract Directory
- Security Manager
- Resource Diagnostics

Developer Resources...

About Microsoft Robotics Studio...

DSS Runtime Fuzzing <<Start without manifest>> Create
Description: Fuzzing service generates rnadom requests and low level packets to test error handling and robustness in the runtime

DSS Runtime HTML Post Form Test <<Start without manifest>> Create
Description: Validates DSS Runtime HTTP Transport support for HTML Post requests

DSS Runtime Test Suite <<Start without manifest>> Create
Description: Suite of tests that test low level functionality of the DSS Runtime

DssService1 <<Start without manifest>> Create
Description: The DssService1 Service

DssService3 <<Start without manifest>> Create
Description: The DssService3 Service

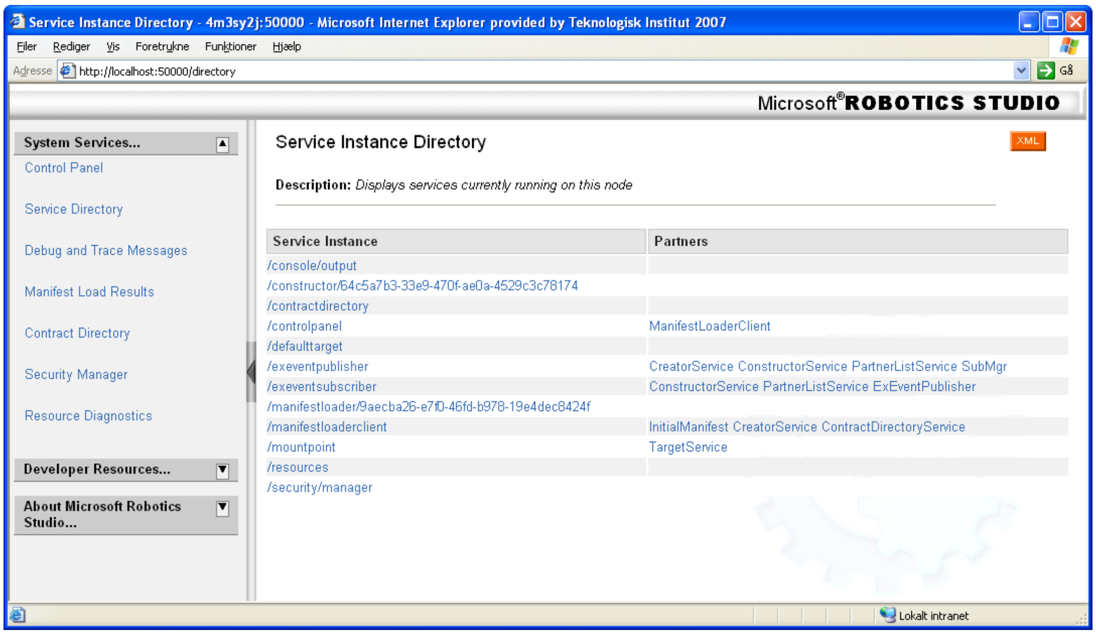
ExEventPublisher <<Start without manifest>> Create
Description: The ExEventPublisher Service
http://4m3sy2j:50000/exeventpublisher Drop

ExEventSubscriber <<Start without manifest>> Create
Description: The ExEventSubscriber Service
http://4m3sy2j:50000/exeventsubscriber Drop

Lokalt intranet

The control panel service lists services in the contract directory, and links are provided to the services running. Services can be started and stopped here.

The service directory


Service Instance Directory - 4m3sy2j:50000 - Microsoft Internet Explorer provided by Teknologisk Institut 2007
Filer Rediger Vis Foretrukne Funktioner Hjælp
Adresse http://localhost:50000/directory
Microsoft® ROBOTICS STUDIO
System Services...
Control Panel
Service Directory
Debug and Trace Messages
Manifest Load Results
Contract Directory
Security Manager
Resource Diagnostics
Developer Resources...
About Microsoft Robotics Studio...
XML
Lokalt intranet

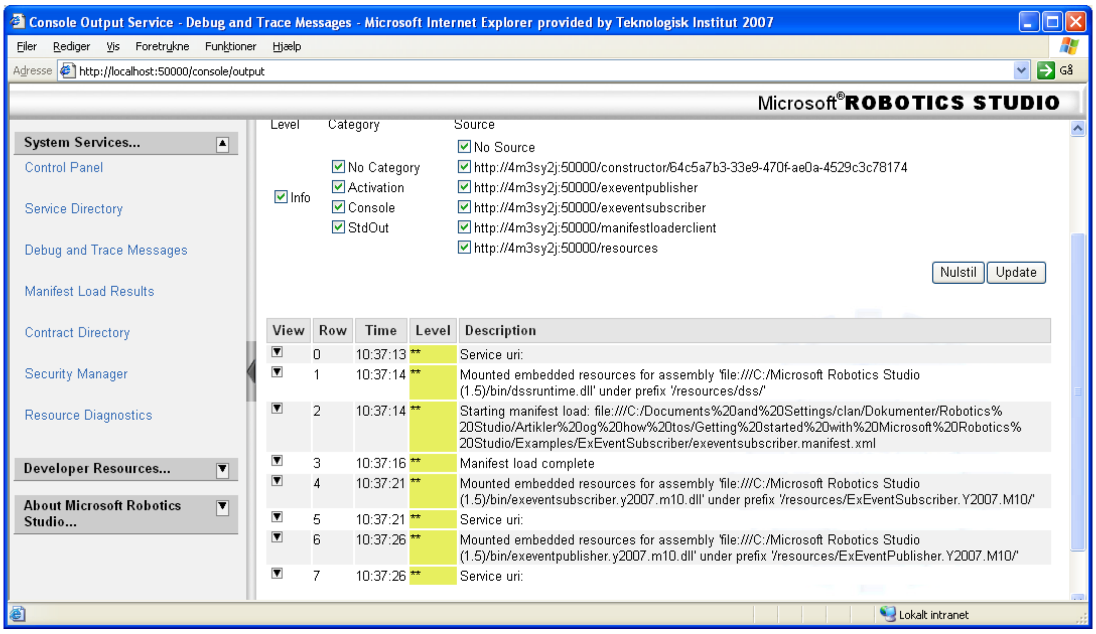
Service Instance Directory

Description: *Displays services currently running on this node*

Service Instance	Partners
/console/output	
/constructor/64c5a7b3-33e9-470f-ae0a-4529c3c78174	
/contractdirectory	
/controlpanel	ManifestLoaderClient
/defaulttarget	
/exeventpublisher	CreatorService ConstructorService PartnerListService SubMgr
/exeventsubscriber	ConstructorService PartnerListService ExEventPublisher
/manifestloader/9aecba26-e7f0-46fd-b978-19e4dec8424f	
/manifestloaderclient	InitialManifest CreatorService ContractDirectoryService
/mountpoint	TargetService
/resources	
/security/manager	

The service directory provides a list of currently running services and their partners.
Direct links are provided for all services.

The debug and trace output


Console Output Service - Debug and Trace Messages - Microsoft Internet Explorer provided by Teknologisk Institut 2007
Filer Rediger Vis Foretrukne Funktioner Hjælp
Adresse http://localhost:50000/console/output
Microsoft® ROBOTICS STUDIO
System Services...
Control Panel
Service Directory
Debug and Trace Messages
Manifest Load Results
Contract Directory
Security Manager
Resource Diagnostics
Developer Resources...
About Microsoft Robotics Studio...
Nulstil
Update
Lokalt intranet

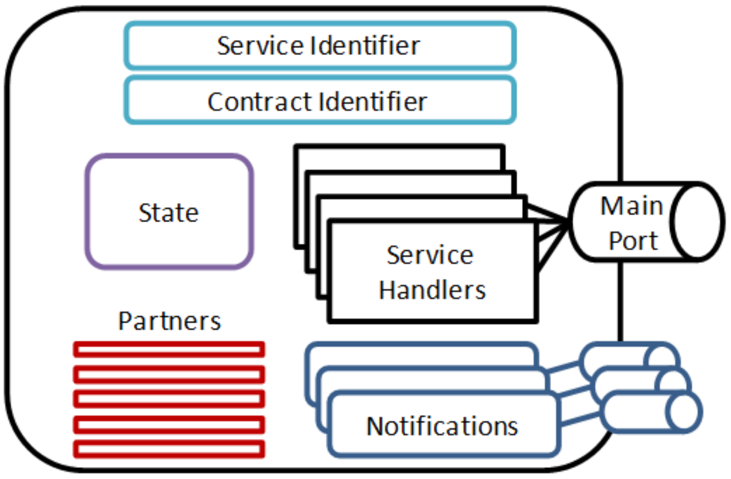
Level	Category	Source
		✓ No Source
	✓ No Category	✓ http://4m3sy2j:50000/constructor/64c5a7b3-33e9-470f-ae0a-4529c3c78174
✓ Info	✓ Activation	✓ http://4m3sy2j:50000/exeventpublisher
	✓ Console	✓ http://4m3sy2j:50000/exeventsubscriber
	✓ StdOut	✓ http://4m3sy2j:50000/manifestloaderclient
		✓ http://4m3sy2j:50000/resources

View	Row	Time	Level	Description
▼	0	10:37:13	**	Service uri:
▼	1	10:37:14	**	Mounted embedded resources for assembly 'file:///C:/Microsoft Robotics Studio (1.5)/bin/dssruntime.dll' under prefix '/resources/dss/'
▼	2	10:37:14	**	Starting manifest load: file:///C:/Documents%20and%20Settings/clan/Dokumenter/Robotics%20Studio/Artikler%20og%20how%20tos/Getting%20started%20with%20Microsoft%20Robotics%20Studio/Examples/ExEventSubscriber/exeventsubscriber.manifest.xml
▼	3	10:37:16	**	Manifest load complete
▼	4	10:37:21	**	Mounted embedded resources for assembly 'file:///C:/Microsoft Robotics Studio (1.5)/bin/exeventsubscriber.y2007.m10.dll' under prefix '/resources/ExEventSubscriber.Y2007.M10/'
▼	5	10:37:21	**	Service uri:
▼	6	10:37:26	**	Mounted embedded resources for assembly 'file:///C:/Microsoft Robotics Studio (1.5)/bin/exeventpublisher.y2007.m10.dll' under prefix '/resources/ExEventPublisher.Y2007.M10/'
▼	7	10:37:26	**	Service uri:

The debug and trace output contains information about services, their load process and state.

The Building Blocks of a MSRS Project

A service contains the following parts:



This is simply a skeleton created by the project wizard. State, service handlers, notifications and partners are what have to be developed to provide the service with its behaviour.

A newly created project consists of a number of files but only two are of interest at this point. These files are *<ServiceName>.cs* and *<ServiceName>Types.cs* where ServiceName is the name you select for your service at the time of creation.

<ServiceName>.cs

This file contains the class that implements the service itself – its behaviour. This is where start-up code and service handlers for interaction with the service are defined. The Get handler is the only handler defined here at this time.

<ServiceName>Types.cs

In this file the following classes are defined:

- The contract (Contract)
- The service state (<ServiceName>State)
- The service's main port (<ServiceName>Operations)
- Each of the service's possible operations (only the Get message is created by the wizard)

At the project creation, the service's contract identifier is set to a default value which should not need any alterations.

The service state class contains all the information that should be available to other services. If it was a camera service for example, this would be where the image would be kept to allow other services to fetch the image data through a Get request.

The operations port set defines the messages which the service can accept. It is performed by using the most fundamental CCR element: a port or in this case a port set, i.e. a collection of ports, one for each message type. This is also called the service's main port as it is used by other services to communicate with this service.

The main port by default supports three different message types but only the Get message is defined here. The two other message types along with their handlers are predefined in the framework.

The GetHandler

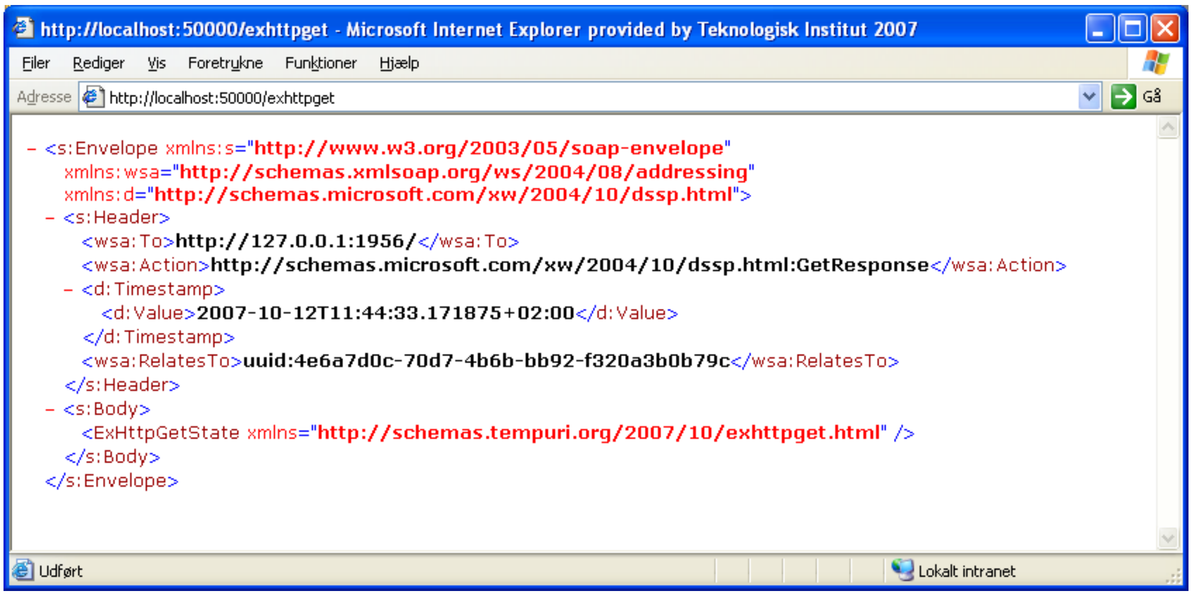
To respond to messages arriving at the main port, service handlers must be defined and associated with the port for each of the message types defined in the operations port set. In a newly created project, this is demonstrated in the most basic form by the GetHandler method in the file <ServiceName>.cs.
The GetHandler method accepts Get objects and is therefore able to extract parameter data from the message. It is also able to respond to the originator of the message trough the response port in the Get message object.
To identify this specific method as the service handler for the main port (and not just some method that happens to accept Get objects), the method must be tagged with the ServiceHandler attribute.
This attribute allows you at the same time to specify whether this method must be run exclusively or if it can be run concurrently with other methods. In general all service handlers perform concurrently unless they modify the service state.

Service handlers must be public virtual methods with IEnumerator<ITask> as the return type. This implies that the method must end with a yield break. It is part of the CCR concurrency model which is outside the scope of this guide.

The default Get handler simply returns the current service state to the message originator by posting the state on the response port in the Get message. A clever feature here is that the Get handler does not know or care whether the originator is on the same host, on another host on the same computer or on a computer connected via a network.

The state is returned as the body of a SOAP envelope. This envelope also includes a number of headers but they are not interesting at this point.

The source code for the examples below can be found in the Examples folder as the project ExHttpGet.
The Get response can be seen by starting the project, opening a web browser and navigating to the services URL. The default URL for the service will be "localhost:50000/<ServiceName>".


http://localhost:50000/exhttpget - Microsoft Internet Explorer provided by Teknologisk Institut 2007
Filer Rediger Vis Foretrukne Funktioner Hjælp
Adresse http://localhost:50000/exhttpget
Gå
Udført
Lokalt intranet

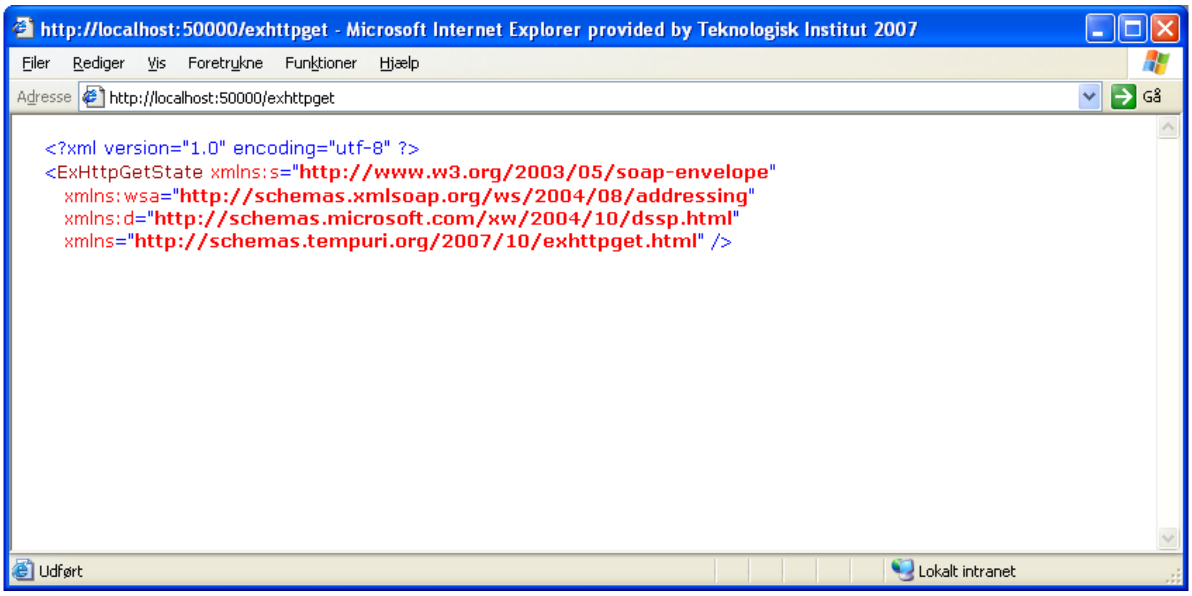
```
- <s:Envelope xmlns:s="http://www.w3.org/2003/05/soap-envelope"
    xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
    xmlns:d="http://schemas.microsoft.com/xw/2004/10/dssp.html">
  - <s:Header>
      <wsa:To>http://127.0.0.1:1956/</wsa:To>
      <wsa:Action>http://schemas.microsoft.com/xw/2004/10/dssp.html:GetResponse</wsa:Action>
    - <d:Timestamp>
        <d:Value>2007-10-12T11:44:33.171875+02:00</d:Value>
      </d:Timestamp>
      <wsa:RelatesTo>uuid:4e6a7d0c-70d7-4b6b-bb92-f320a3b0b79c</wsa:RelatesTo>
    </s:Header>
  - <s:Body>
      <ExHttpGetState xmlns="http://schemas.tempuri.org/2007/10/exhttpget.html" />
    </s:Body>
  </s:Envelope>
```

This is the information presented by Internet Explorer when looking at the ExHttpGet service that has just been created. Depending on the web browser used, it may be possible to see something a little different. The important thing here is the body section and in particular the ExHttpGetState field. This is empty here because the service state class does not yet contain any members.

Responding to HTTP Get

The Get response shown above is intended for service to service communication. Usually, it is retrieving state information via a web browser and the response should be in a human readable form.
To achieve this, you must implement the HTTP Get message handler.

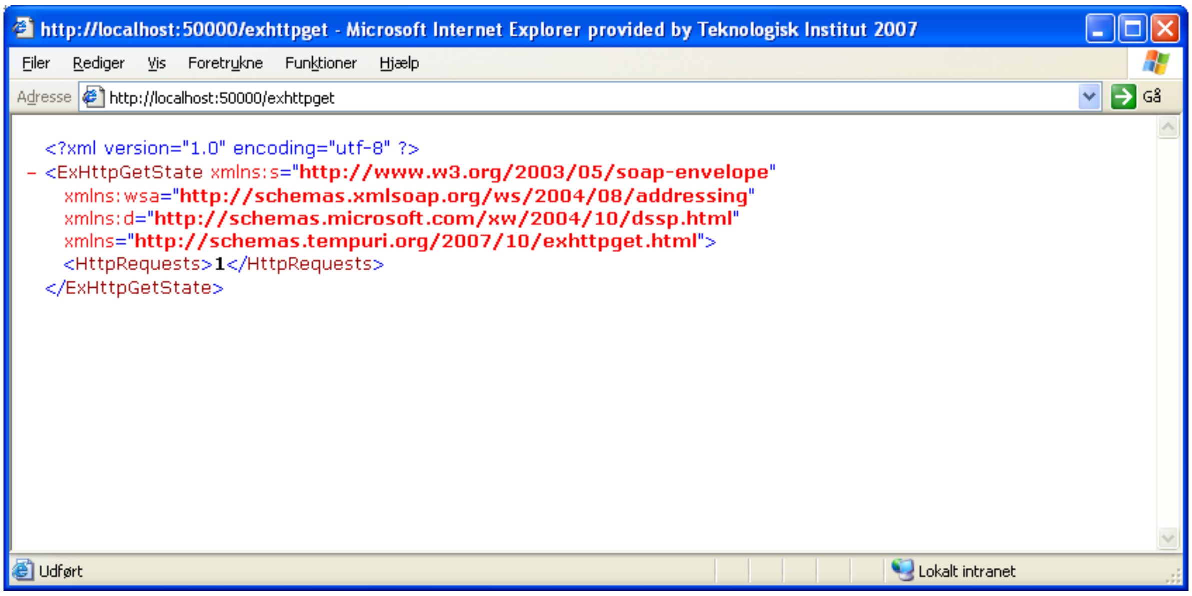
By implementing a HTTP Get handler that simply returns the same state object as before, a somewhat more readable output is provided as HTTP does not rely on SOAP envelopes.


http://localhost:50000/exhttpget - Microsoft Internet Explorer provided by Teknologisk Institut 2007
Filer
Rediger
Vis
Foretrukne
Funktioner
Hjælp
Adresse
http://localhost:50000/exhttpget
Gå
Udført
Lokalt intranet

```
<?xml version="1.0" encoding="utf-8" ?>
<ExHttpGetState xmlns:s="http://www.w3.org/2003/05/soap-envelope"
  xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
  xmlns:d="http://schemas.microsoft.com/xw/2004/10/dssp.html"
  xmlns="http://schemas.tempuri.org/2007/10/exhttpget.html" />
```

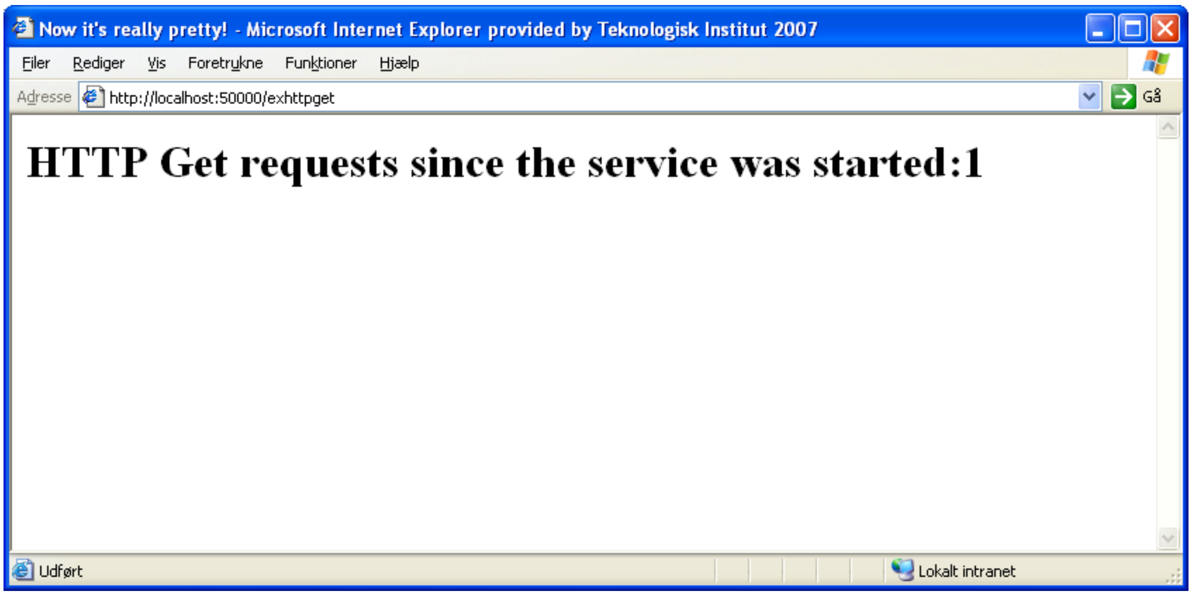
The state information along with a few XML attributes is displayed this time only.

If we add a single data member of type integer to the state class, the following result may be achieved.


http://localhost:50000/exhttpget - Microsoft Internet Explorer provided by Teknologisk Institut 2007
Filer Rediger Vis Foretrukne Funktioner Hjælp
Adresse http://localhost:50000/exhttpget
Gå
Udført
Lokalt intranet

```
  <?xml version="1.0" encoding="utf-8" ?>
- <ExHttpGetState xmlns:s="http://www.w3.org/2003/05/soap-envelope"
    xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
    xmlns:d="http://schemas.microsoft.com/xw/2004/10/dssp.html"
    xmlns="http://schemas.tempuri.org/2007/10/exhttpget.html">
    <HttpRequests>1</HttpRequests>
  </ExHttpGetState>
```

It still requires a developer to make sense out of it, to take the final step and to actually make it readable. XSL transformations must be used.


Now it's really pretty! - Microsoft Internet Explorer provided by Teknologisk Institut 2007
Filer Rediger Vis Foretrukne Funktioner Hjælp
Adresse http://localhost:50000/exhttpget
Gå
Udført
Lokalt intranet

HTTP Get requests since the service was started:1

You now have all the XSLT expressions at your disposal to present the state information in any way you like.

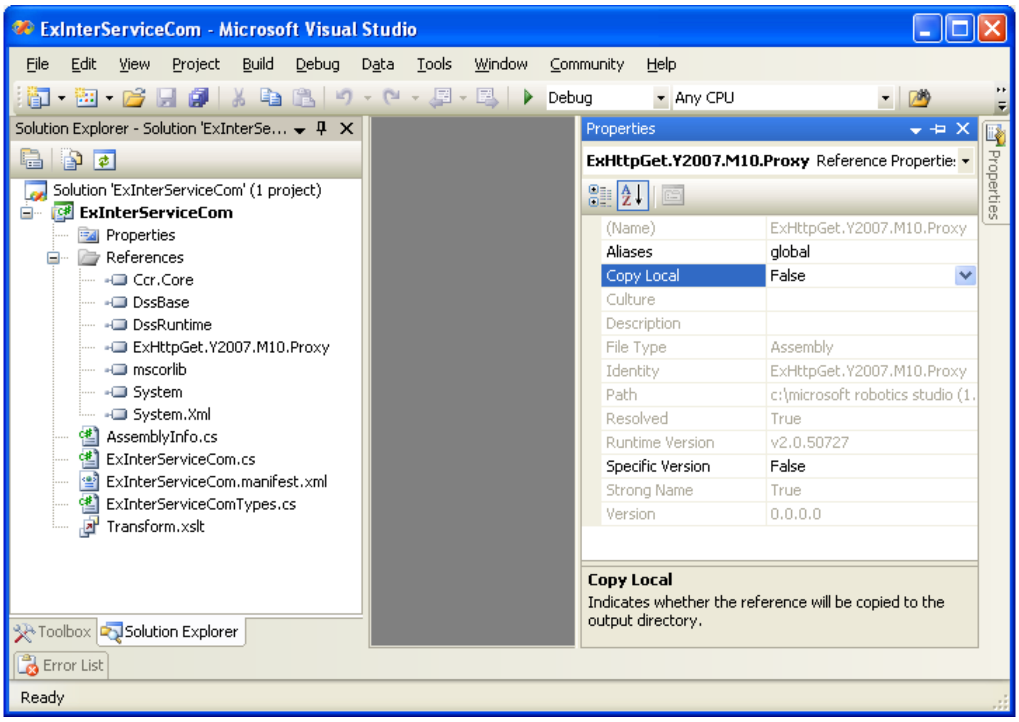
Communicating with Other Services

The Get message and other DSSP messages are used to communicate between services. Since the service that was just created already supports DSSP Get (this was implemented by the wizard), we can now create another service[3] which - upon a HTTP Get request - copies the state information from the first service via a DSSP Get and sends it as its HTTP Get response.

[3] This is the ExInterServiceCom project.

Some knowledge of the services must be shared to enable this communication, and it is performed via proxies and is called partnering. When a service is built, a proxy is automatically generated. This proxy contains information on classes in the service, the main port, the contract etc.

In order to import the information in a new project, a reference should be added to the proxy (it can be found in projects directory under /Proxy/obj/Debug/) in the project manager. Set its “Copy local” property to “false” as we only need a reference and not a copy of it. Also set its “Specific Version” property to false to allow changes to referenced service.


ExInterServiceCom - Microsoft Visual Studio
File Edit View Project Build Debug Data Tools Window Community Help
Debug
Any CPU
Solution Explorer - Solution 'ExInterSe...
Properties
Toolbox
Solution Explorer
Error List
Ready

Solution 'ExInterServiceCom' (1 project)
- ExInterServiceCom
  - Properties
  - References
    - Ccr.Core
    - DssBase
    - DssRuntime
    - ExHttpGet.Y2007.M10.Proxy
    - mscorlib
    - System
    - System.Xml
  - AssemblyInfo.cs
  - ExInterServiceCom.cs
  - ExInterServiceCom.manifest.xml
  - ExInterServiceComTypes.cs
  - Transform.xslt

ExHttpGet.Y2007.M10.Proxy Reference Properties

(Name)	ExHttpGet.Y2007.M10.Proxy
Aliases	global
Copy Local	False
Culture	
Description	
File Type	Assembly
Identity	ExHttpGet.Y2007.M10.Proxy
Path	c:\microsoft robotics studio (1.
Resolved	True
Runtime Version	v2.0.50727
Specific Version	False
Strong Name	True
Version	0.0.0.0

Copy Local
Indicates whether the reference will be copied to the output directory.

In order to post a message to the remote service, an instance of the remote services main port type should be created and attributed to the Partner attribute whereby it can be connected with the actual remote main port. Now simply post messages to the port and the DSS will forward it to the correct host and service.

Notifying Other Services of State Changes

The previous example requests the state information every time it is required. It may result in many requests returning the exact same information. To avoid this superfluous traffic, the next example[4] uses event notification where the information provider (the event publisher) notifies the information consumer (the subscriber of the event) whenever the state information has changed. The consumer then caches the new state.

Event notification requires slight modifications in both provider and consumer services.

The provider requires the following steps to support subscription to event notification:
- Partner with the DSS’ built in subscription manager, i.e. get a handle to its main port.
- Define a subscribe message type and add it to the services main port.

[4] The projects ExEventPublisher and ExEventSubscriber.

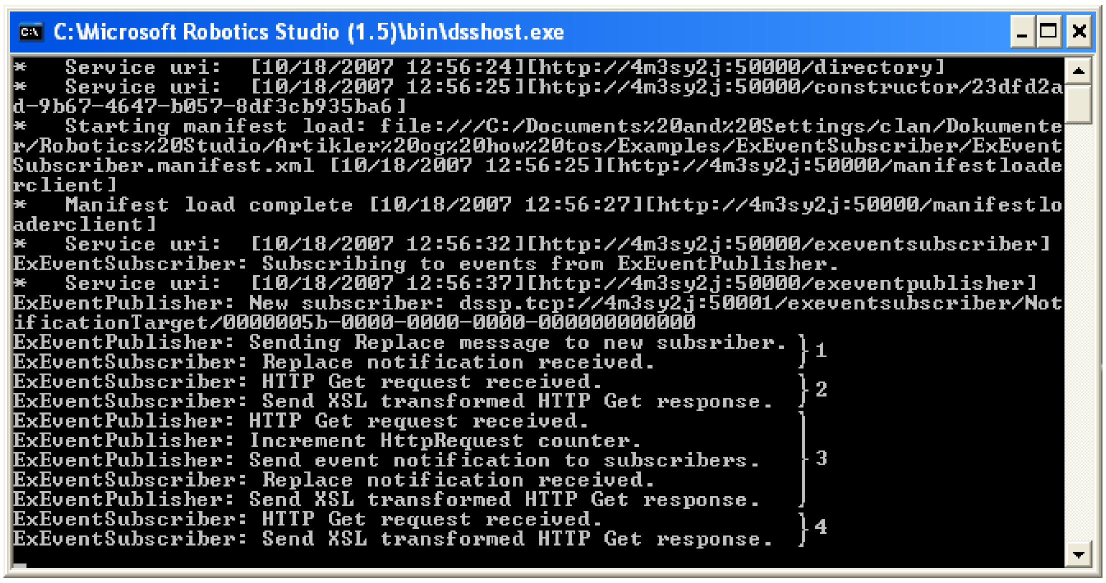
- Implement a handler for the subscribe message type.

A message must subsequently be forwarded to the subscription manager each time the service state is modified. The subscription manager then forwards the message to all registered event subscribers.

The steps required for the subscriber are equally simple (assuming a partnership is already established with the remote service):

- An additional port of the remote services main port type must be created (but not partnered with the remote service).
- A subscribe message must be sent to the remote service indicating that events must be sent to the newly created port.
- Message handlers must be created for each message type that will be forwarded, and the handlers must be associated with the port.

Starting the project ExEventSubscriber provides the following console output:



```
*   Service uri:  [10/18/2007 12:56:24][http://4m3sy2j:50000/directory]
*   Service uri:  [10/18/2007 12:56:25][http://4m3sy2j:50000/constructor/23dfd2a
d-9b67-4647-b057-8df3cb935ba6]
*   Starting manifest load: file:///C:/Documents%20and%20Settings/clan/Dokumente
r/Robotics%20Studio/Artikler%20og%20how%20tos/Examples/ExEventSubscriber/ExEvent
Subscriber.manifest.xml [10/18/2007 12:56:25][http://4m3sy2j:50000/manifestloade
rclient]
*   Manifest load complete [10/18/2007 12:56:27][http://4m3sy2j:50000/manifestlo
aderclient]
*   Service uri:  [10/18/2007 12:56:32][http://4m3sy2j:50000/exeventsubscriber]
ExEventSubscriber: Subscribing to events from ExEventPublisher.
*   Service uri:  [10/18/2007 12:56:37][http://4m3sy2j:50000/exeventpublisher]
ExEventPublisher: New subscriber: dssp.tcp://4m3sy2j:50001/exeventsubscriber/Not
ificationTarget/0000005b-0000-0000-0000-000000000000
ExEventPublisher: Sending Replace message to new subsriber.  } 1
ExEventSubscriber: Replace notification received.           }
ExEventSubscriber: HTTP Get request received.                } 2
ExEventSubscriber: Send XSL transformed HTTP Get response.   }
ExEventPublisher: HTTP Get request received.                 }
ExEventPublisher: Increment HttpRequest counter.             }
ExEventPublisher: Send event notification to subscribers.    } 3
ExEventSubscriber: Replace notification received.            }
ExEventPublisher: Send XSL transformed HTTP Get response.    }
ExEventSubscriber: HTTP Get request received.                } 4
ExEventSubscriber: Send XSL transformed HTTP Get response.   }
```

- Host and service initialization ends with section number 1.
- Then (section number 2) the event subscriber service receives a HTTP Get request to which it sends an XSL transformed response.
- Section 3 covers a HTTP Get request to the event publisher service which causes an increment of the count that in turn generates an event. This event is received by the event subscriber.
- The final section (number 4) is another HTTP Get to the event subscriber reflecting the new value on the web browser.

Legacy Drivers

Migrating to Microsoft Robotics Studio is quite a step as compared to writing classic single threaded code. Fortunately it is not difficult to convert old drivers to fit into MSRS once you know the basics of a MSRS project.

As an example let us consider an existing driver for the SRV-1 robot from Surveyor Corporation[5].

[5] For more information see www.surveyor.com.

The driver project can be found in the example folder as the project SrvDriver and the project SrvDemonstrator is a demonstration project putting the driver service to use. The driver class taken from an existing C# project is the SRVLowLevelDriver.cs file. This driver was written for a single threaded application - and as this will often be the case with old drivers - the example represents a classical migration path when first starting to use MSRS.

The driver uses a serial port to communicate with the SRV-1 robot. Each transaction consists of one or more bytes forwarded to the robot. A short delay will occur while the robot processes the command and finally a response will show up. Transactions are not allowed to overlap i.e. the full response of a command must be received before a new command is forwarded. Since a service may be used by several other services, a time transaction atomicity must be guaranteed by the driver service. The CCR provides the perfect method for this through its use of ports and handlers.

The port mechanism used to communicate with a service (the main port) is replicated within the service to provide the same port-based interface to the driver as to a service.

The handlers that are triggered by messages arriving at a port can be configured to run only if no other handlers are running for that port. Here it is used to give the atomicity required. All message handlers on the internal port are scheduled to run exclusively if a message arrives while one is already being processed. It is then queued and processed when the previous one is completed.

Sending a command to the robot and receiving a reply is no longer a call to a method in the robot driver but simply a message posted to the internal command port and the subsequent event notification.

This solution requires that a call to one of the driver-methods does not return until the action is completed. If a method relies on call-backs or the APM, the entire transaction must be encapsulated into one synchronous method. The CaptureImage method in SRVLowLevelDriver.cs is an example of such an encapsulating method.

To finally be able to forward commands to the robot from a remote service through the driver service, the service itself must accept robot control messages and forward them to the internal command port. This can be executed by forwarding the specific robot command as the body of a robot control message which is subsequently accepted trough the services main port. The service handler for the carrier message then forwards a message on the internal port based on the body of the received message. Alternatively each robot command can be accepted on the main port and simply forwarded directly by their service handlers.
The SRVDriver uses the latter technique.

The messages forwarded to the internal command port lead to changes in the driver's state which in turn result in events being generated.
There are three types of events:

- Event message contains and replaces the complete state information. High network load but easy to implement.
- Event message contains only the changed state information. Leads to more complex code but may result in reduced network traffic.
- Event message contains no state information. If state information is required it must be fetched manually afterwards. If a large number of events are expected before the state information is required, this method may significantly reduce network traffic.

Use whichever type of event is most appropriate in the situation at hand.

Most legacy hardware drivers should be easy to migrate to MSRS by using the technique described above. For instance the driver used in the example above required only one modification (the encapsulating method) to work with MSRS.

What's Next

The examples here are only a scratch in the surface of MSRS but it demonstrates the very few steps required to get a project up and running.
All in all Microsoft Robotics Studio provides a solid and easy to use framework for creating robotics applications. CCR and DSS may find more widespread use but it certainly has its justification in robotics. DSS may be for heavy weight for some applications and as both the CCR and the DSS requires the .Net framework to run, a significant volume of system resources are required.

About the author

Claus Andersen is a student programmer at the Danish Technological Institute. Claus studies ICT engineering (specialization in Embedded Software) at the Engineering College of Aarhus.